AD-A269 945

REPORT DOCUMENTATION PAGE

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED FINAL/01 DEC 91 TO 31 MAR 93
4. TITLE AND SUBTITLE UUGM CODE DEVELOPMENT (U)		5. FUNDING NUMBERS
6. AUTHOR(S) Professor S. Eidelman		6775/DARPA F49620-89-C-0087
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Science Applications International Corp 1710 Goodridge Drive, MS 2-3-1 McLean VA 22102	AEOSR-TR-	8. PERFORMING ORGANIZATION REPORT NUMBER 2
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFOSR/NM 110 DUNCAN AVE, SUITE B115 BOLLING AFB DC 20332-0001		10. SPONSORING/MONITORING AGENCY REPORT NUMBER F49620-89-C-0087
11. SUPPLEMENTARY NOTES		
12a. DISTRIBUTION/AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE: DISTRIBUTION IS UNLIMITED		12b. DISTRIBUTION CODE UL

13. ABSTRACT (Maximum 200 words)

The primary objective of SAIC was to develop an unstructured grid algorithm and code that dynamically adapts to the computed solution of the time dependent Euler equations of gasdynamics in two and three spatial dimensions. Important requirements that were imposed on the algorithm were: robustness, accuracy, efficiency, flexibility, and adaptability. The main research and code development effort was focused on achieving these objectives; extensive testing and code validation effort was undertaken to demonstrate the code's performance for realistic CFD problems. The method is accurate in all flow regimes from subsonic to hypersonic. The main achievement was the development of the AUGUST code (Adaptive Unstructured Grid Upwind Second Order for Triangles). AUGUST is implemented for solution of Euler's equations on dynamically adaptive triangular or tetrahedral grids. The code fully implements the Second-Order Godunov method, allowing accurate and robust numerical solution of Euler equations of gas dynamics. A new method was developed for Direct Dynamic Grid Refinement (DDR). The AUGUST code was also implemented for multiphase, multicomponent flows. A combined structured/unstructured version of the AUGUST code was also developed. The AUGUST code was extensively validated for a wide range of problems and has proven to be a robust tool.

93-22349

14. SUBJECT TERMS			15. NUMBER OF PAGES 255
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT SAR(SAME AS REPORT)

UUGM CODE DEVELOPMENT

Science Applications International Corporation
An Employee-Owned Company

UUGM CODE DEVELOPMENT

Science Applications International Corporation
An Employee-Owned Company

Accesion For		
NTIS CRA&I	X	
DTIC TAB	☐	
Unannounced	☐	
Justification		
By		
Distribution /		
Availability Codes		
Dist	Avail and / or Special	
A-1		

Post Office Box 1303, 1710 Goodridge Drive, McLean, Virginia 22102 (703) 821-4300

UUGM CODE DEVELOPMENT

SAIC Final Report #SAIC-93/1152

Final Report for work accomplished under
AFOSR Contract #F49620-89-C-0087 during period
15 October 1990 through 30 November 1992

Contributors:
Shmuel Eidelman
William Grossmann
Isaac Lottati
Xiaolong Yang
Marty Fritts
Adam Drobot
Michael Kress
Aaron Friedman

Prepared by:
Science Applications International Corporation
Applied Physics Operation
1710 Goodridge Drive, MS 2-3-1
McLean, VA 22102

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. 6775
Monitored by AFOSR Under Contract No. F49620-89-C-0087

July 26, 1993

EXECUTIVE SUMMARY

This progress report documents the effort conducted at SAIC from 15 October 1990 through 31 May 1993, under DARPA and AFOSR contract #F49620-89-C-0087 entitled "UUGM Code Development".

Scope of Research

The primary objective of SAIC was to develop an unstructured grid algorithm and code that dynamically adapts to the computed solution of the time dependent Euler equations of gasdynamics in two and three spatial dimensions. Important requirements that were imposed on the algorithm were: robustness, accuracy, efficiency, flexibility, and adaptability. The main research and code development effort was focused on achieving these objectives; extensive testing and code validation effort was undertaken to demonstrate the code's performance for realistic CFD problems. The method is accurate in all flow regimes from subsonic to hypersonic.

Achievements

The main achievement was the development of the AUGUST code (Adaptive Unstructured Grid Upwind Second Order for Triangles). AUGUST is implemented for solution of Euler's equations on dynamically adaptive triangular or tetrahedral grids. The code fully implements the Second-Order Godunov method, allowing accurate and robust numerical solution of Euler equations of gas dynamics.

A new method was developed for Direct Dynamic Grid Refinement (DDR). This method allows grid refinement in arbitrary regions of the computational domain, using only one level of undirectness in the logical data structure. The DDR is an integral part of the AUGUST solver and allows manipulation of the grid as a part of the solution. The adapted grid is not only more refined in the adaptation regions of the flow but is also improved structurally due to a refinement algorithm.

The AUGUST code was also implemented for multiphase, multicomponent flows. We used a multiple-fluid description, where a separate set of conservation laws is used to describe every flow component. In our approach Lagrangian tracers are used to describe sparse or discrete flow components that do not form a continuum. Use of unstructured triangular grids allows adjustment of the grid resolution to the accuracy requirements in the flow subdomains.

A combined structured/unstructured version of the AUGUST code was also developed. Following this approach the unstructured adaptive grid is used only in the flow regions requiring adaptation or description of the complex geometry elements. The structured grid is used to simulate the larger part of the computational domain. This approach has allowed us to capitalize on the advantages of both structured and unstructured grid approaches. Using the structured/unstructured grid version of the

AUGUST code, we simulated the shock wave focusing problem for the reflector used for extracorporeal shock-wave lithotripsy. In this simulation, we showed that the solution smoothly transits through the interfaces between the grids, maintaining the same accuracy and resolution.

The AUGUST code was extensively validated for a wide range of problems and has proven to be a robust tool. The code was initiated at the start of the UUGM project and has now evolved into a production code that is used for many applied problems. The list of applications includes potential flow past an ellipse, hypersonic flow past a flat plate, shock diffraction over single and double wedges, mine explosions under vehicles, pulsed detonation engines, shock focusing in air, and nonideal airburst in multiphase media. The code has shown the required robustness and insensitivity to the initial user specified grid. The number of nodes required to obtain a high-quality solution is significantly smaller than for structured grid codes. This is particularly true for transient problems with complicated flows having discontinuities.

It is important to note that the AUGUST code obtains a high resolution solution with no "knobs." The various flow regimes, except those requiring a different definition of boundary and initial conditions, were simulated using the same code.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

1. INTRODUCTION

1.1 RECENT CFD DEVELOPMENT

Computational fluid dynamics (CFD) development over the past twenty years has truly been outstanding. The recent CFD developments that are particularly important are: 1) advances in flow solvers in all the regimes of fluid flow (very low speed and subsonic flows, transonic flow, supersonic and hypersonic flows), 2) advances in unstructured adaptive gridding techniques and, 3) advances in chemical and particle kinetic modeling for fluid flows. Developments in graphics and visualization, construction of graphical user interfaces (GUIs) and advances in large database management have also played an important role in the scale and complexity of problems that can now be realistically simulated by CFD techniques. SAIC has been involved in all aspects of these developments and is on the forefront of CFD technology development.

DARPA, NASA, DNA and DOE have for the most part been the largest benefactors of CFD development, and each agency today is actively pursuing CFD applications to real problems. Full 3-D unsteady flows about military and commercial aircraft are routinely simulated to assess aerodynamic performance characteristics, and where it used to require several hundred hours of CRAY CPU time it now takes minutes to an hour on a supercomputer or a like time on workstations, depending on the specifics of the problem being solved. The U.S. Marine Corps' latest initiative in the development of blast (due to land mines) resistant vehicles is being pursued successfully with the aid of full 3-D CFD simulations of land mine blast effects on truck configurations. The CFD technology developed in SAIC's UUGM contract is playing a leading role in this Marine Corps effort (see Section 3.4). Many other such examples of improvements in CFD performance exist. In view of this, it is quite appropriate to begin to transition CFD technology into other disciplines that can take advantage of realistic CFD based simulation.

1.2 UNSTRUCTURED MESHES IN COMPLEX GEOMETRIES

Current emphasis in CFD calls for solutions of applied physical problems for complex realistic geometries.[1] In addition to the inherent difficulties in describing the details of the complex three-dimensional geometry, the flow fields usually have an inhomogeneous structure. Regions of rapid change of the flow functions and chemical reactions will be embedded in regions where the flow gradients are relatively small. Accurate simulations of flows in regions with strong gradients is key to the overall accuracy of physical, chemical and biological simulations. For this reason most of the software and hardware computational resources are defined by the accuracy requirements of these flow regions and geometry of the computational domain.

Early CFD research was almost entirely concerned with the formulation the mathematical models of the flow and methods of solution. Mesh generation was regarded as secondary and meshes were developed for specific cases. During this early period very

significant improvements were made in the methods of integration of the partial differential equations of gasdynamics. Presently, as a result of steady improvement in the various integration techniques, the advantages which could be gained by using better flow solvers have become limited. On the other hand substantial progress is anticipated in the areas of grid generation and algorithm development.[2]

Currently, most numerical simulations employ structured meshes composed of quadrilaterals in two dimensions or hexahedra in three dimensions. However, it has become evident that the quadrilateral structured grids cannot satisfy the requirements of large scale numerical simulations over complex geometries in three dimensions. The physics of the flow about a complete aircraft is extremely complex. Yet the flow in many distinct regions and regimes may be represented by fairly well-known physical theories. Vortices shed by lifting surfaces are confined to fairly thin wake regions. Exhaust plumes can be initially approximated by regular bounding surfaces. Flow disturbances due to shocks are confined to thin discontinuities. Boundary layers are restricted to near-wall regions. Each of these flow regions requires different theories, different resolution and different numerical algorithms. This diversity of computational requirements cannot be satisfied by the quadrilateral structured grids.

Recently proposed alternatives to quadrilateral grids use triangles in two dimensions and tetrahedra in three dimensions. For these grids the mesh will generally lose its structure, allowing a new degree of flexibility in treating complex geometries.[3,4] Unstructured grids can relatively easily be adapted to follow flow features, thereby increasing the solution accuracy. The result has been the development of adaptive refinement techniques which have been used with great success for two dimensional simulations on unstructured triangular grids. These methods have resulted in the resolution of previously difficult details in the evolving flows over complex configurations.

However, it is not a trivial task to adapt this approach to three-dimensional simulations. One of the problems is the generation of the adaptive grid. Since the grid is constructed from the volume elements (tetrahedra) the moving front is made up of a surface of triangular faces. It should be noted that this moving front can and will change its shape during the computation as time evolves. It is necessary to take care when determining the intersections of planar faces, and to ensure that no overlapping of tetrahedra occurs.

2. UUGM: UNIVERSAL CFD SIMULATION ENVIRONMENT

The Ultimate Unstructured Grid Method (UUGM) represents a new approach to the computational domain discretization. The principal advantage of the method is most apparent for simulations of complicated flow regimes with physical and chemical processes over bodies with complex geometries in three dimensions.

The usual technique employed in regridding is called hierarchical dynamic refinement (H refinement). The idea here is to retain a history of the original grid and the

subdivisions needed to change it into the current grid, so that it is always possible to retrace these steps and get back to previous grids. While this feature is useful in modeling reversible processes, it is generally unnecessary, and it increases overhead costs. Our implementation (Direct Dynamic Refinement) is Markovian, in the sense that the way regridding is done depends only on the current grid and flow conditions.

The other distinguishing feature is the use of the Second-order Godunov method to solve the Euler equations of gasdynamics. The philosophy behind it is to treat the local values of the dependent variables at every point on the grid as initial conditions for a Riemann problem, and to use the resultant solution of that problem to calculate the fluxes of material, momentum, and energy from one cell to the rest. Previous implementations of this method were confined to structured meshes.

2.1 MATHEMATICAL MODEL AND INTEGRATION ALGORITHM

We consider a system of two-dimensional Euler equations written in conservation law form as

$$\frac{\partial U}{\partial t}+\frac{\partial F}{\partial x}+\frac{\partial G}{\partial y}=0, \qquad (2.1)$$

where

$$U=\begin{vmatrix}\rho\\ \rho u\\ \rho v\\ e\end{vmatrix},\quad F=\begin{vmatrix}\rho u\\ \rho u^2+p\\ \rho u v\\ u(e+p)\end{vmatrix},\quad G=\begin{vmatrix}\rho v\\ \rho u v\\ \rho v^2+p\\ v(e+p)\end{vmatrix}.$$

Here u, v are the x, y velocity vector components, p is the pressure, ρ is the density and e is total energy of the fluid. We assume that the fluid is an ideal gas and the pressure is given by the equation of state,

$$p=(\gamma-1)\left[e-\left(\frac{\rho}{2}\right)(u^2+v^2)\right], \qquad (2.2)$$

where γ is the ratio of specific heats and is typically taken as 1.4 for air. It is assumed that an initial distribution of the fluid parameters is given at $t = 0$, and the boundary conditions defining a unique solution are specified for the computational domain.

The system of governing equations in Eq. (2.1) can be written as

$$\frac{\partial U}{\partial t}+\nabla\cdot Q=0, \qquad (2.3)$$

where Q represents the convective flux vector. Integrating Eq. (2.3) over space and using Gauss' theorem produces the expression

$$\frac{\partial}{\partial t}\int_{\Omega} U\, dA + \oint_{\partial\Omega} Q \cdot d\boldsymbol{l} = 0, \tag{2.4}$$

where $d\boldsymbol{l} = \boldsymbol{n}dl$, $\boldsymbol{n}$ is the unit normal vector in the outward direction, and $d\boldsymbol{l}$ is the element of length on the boundary of the domain. Here Ω is the domain of computation and $\partial\Omega$ is the boundary of this domain.

We seek a solution to the system of Eq. (2.1) in the computational domain, which is decomposed in part into triangles with arbitrary connectivity and in part into rectangles using a logically structured grid. We use the advantage of the unstructured grid (Refs. 5-8) to describe the curved boundary of the computational domain and areas that need increased local resolution; this covers a small part of the total computational domain. The largest area of the computational domain is decomposed by the structured grid. The numerical technique for solving Euler's equation on an unstructured grid is described in Refs. 9-11, and the technique for the structured grid is described in Ref. 9. These numerical techniques apply some of the ideas that were introduced in Refs. 13-14. The structured and unstructured codes apply the center-based formulation, i.e., the primitive variables are defined in the center of the cell, which makes the cell the integration volume, while the fluxes are computed across the edges of the cell. The basic algorithmic steps of the Second-Order Godunov method can be defined as follows:

1. Find the value of the gradient at the baricenter of the cell for each gasdynamic parameter U_i;

2. Find the interpolated values of U at the edges of the cell using the gradient values;

3. Limit these interpolated values based on the monotonicity condition;[13]

4. Subject the projected values to the characteristics constraints;[14]

5. Solve the Riemann problem applying the projected values at the two sides of the edges;

6. Update the gas dynamic parameter U according to the conservation equations (1) applying to the fluxes computed and the current timestep.

As was recommended in Ref. 11, we prefer the version based on triangle centers over the vertex-based version of the code. For the same unstructured grid, a center-based algorithm will result in smaller control volumes than a vertex-based. In addition, for the Second-Order Godunov solver, implementation of the boundary conditions is more straightforward and accurate for the center-based algorithm than in the vertex-based version. These two factors, along with the effects of grid connectivity, strongly affect the algorithm accuracy and performance and are the main reasons for the superiority of the center-based version over the vertex version.

Equation (2.4) can be discretized for each element (cell) in the domain

$$\frac{(U_i^{n+1} - U_i^n)}{\Delta t} A_i = \sum_{j=1}^{M} Q_j^n \cdot n_j \, \Delta L_j \, , \tag{2.5}$$

where A_i is the area of the cell; Δt is the marching timestep; U_i^n and U_i^{n+1} are the primitive variables at the center of the cell at time n and at the updated $(n + 1)$st timestep; Q_j is the value of the fluxes across the boundaries on the circumference of the cell where n_j is the unit normal vector to the boundary edge j, and ΔL_j is the length of the boundary edge j. The fluxes Q_j^n are computed applying the Second-Order Godunov algorithm, and Eq. (2.5) is used to update the physical primitive variables u_i according to computed fluxes for each marching timestep Δt. The marching timestep is subjected to the Courant-Friedrichs-Lewy (CFL) constraint.

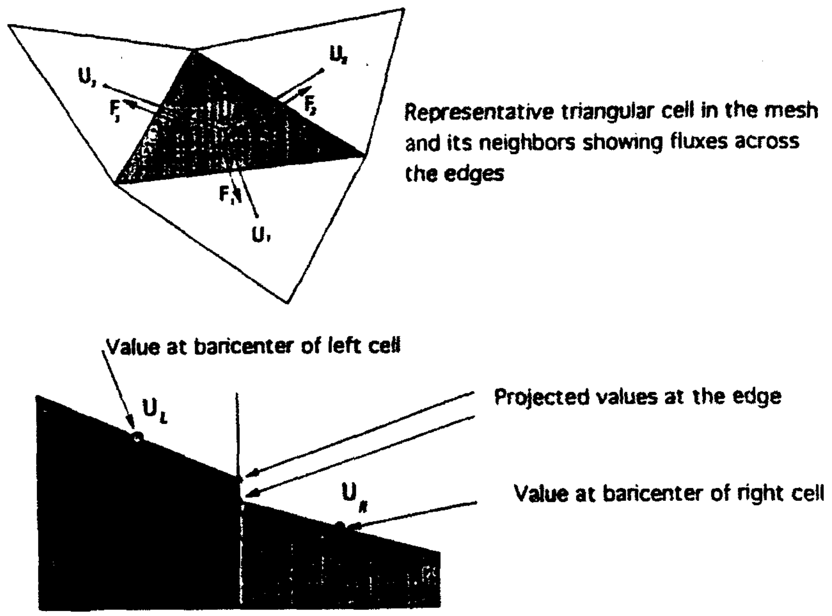


Figure 2.1.1 Representative triangular cell in the mesh showing fluxes and projected values

To obtain second-order spatial accuracy, the gradient of each primitive variable is computed in the baricenter of the cell. This gradient is used to define the projected values of primitive variables at the two sides of the cell edge, as shown in Fig. 2.1.1. The gradient is approximated by a path integral

$$\int_{\Omega} \nabla U_i^{\text{cell}} \, dA = \oint_{\partial\Omega} U_j^{\text{edge}} \, dl \, . \tag{2.6}$$

The notation is similar to the one used for Eq. (2.5), except that the domain Ω is a single cell and U_i and U_j are values at the baricenter and on the edge respectively. The gradient is estimated as

$$\nabla U_i^{\text{cell}} = \frac{1}{A} \sum_{j=1}^{3} U_j^{\text{edge}} \, n_j \, \Delta L_j \tag{2.7}$$

where U_j^{edge} is an average value representing the value o. ,..mitive variable for edge j.

The gradients that are computed at each baricenter are used to project values for the two sides of each edge by piecewise linear interpolation. The interpolated valu..s are subjected to monotonicity constraints.[3] The monotonicity constraint ensures : .it the interpolated values do not create new extrema.

The monotonicity limiter algorithm can be written in the following form

$$U_{\text{proj}}^{\text{edge}} = U_i^{\text{cell}} + \phi \nabla U_i \cdot \Delta \boldsymbol{r} \tag{2.8}$$

where $\Delta \boldsymbol{r}$ is the vector from the baricenter to the point of intersection of the edge with the line connecting the baricenters of the cells over the two sides of this edge. Here ϕ is the coefficient that limits the gradient ∇U_i.

First we compute the maximum and minimum values of the primitive variable in the ith cell and its three neighboring cells that share common edges (see Fig. 2.1.1):

$$\left.\begin{aligned} U_{\text{cell}}^{\max} &= \max\left(U_k^{\text{cell}}\right) \\ U_{\text{cell}}^{\min} &= \min\left(U_k^{\text{cell}}\right) \end{aligned}\right\} k = i, 1, 2, 3. \tag{2.9}$$

The limiter can be defined as

$$\phi = \min\left\{1, \phi_k^{lr}\right\}, k = 1,2,3 \tag{2.10}$$

where the superscript lr stands for left and right of the three edges (6 combinations altogether); ϕ_k^{lr} is defined by

$$\phi_k^{lr} = \frac{\left[1 + \text{sgn}\left(\Delta U_k^{lr}\right)\right] \Delta U_{\text{cell}}^{\max} + \left[1 - \text{sgn}\left(\Delta U_k^{lr}\right)\right] \Delta U_{\text{cell}}^{\min}}{2 \Delta U_k^{lr}}, \quad k = 1, 2, 3 \tag{2.11}$$

where $\Delta U_k^{lr} = \nabla U_i^{lr} \cdot \Delta \boldsymbol{r}_k$ and

$$\left.\begin{aligned} \Delta U_{\text{cell}}^{\max} &= U_{\text{cell}}^{\max} - U_i^{\text{cell}} \\ \Delta U_{\text{cell}}^{\min} &= U_{\text{cell}}^{\min} - U_i^{\text{cell}} \end{aligned}\right\}. \tag{2.12}$$

To obtain second-order accuracy in time and space, we subject the projected values of the left and right side of the cell edge to characteristic constraints following Ref. 4. The one-dimensional characteristic predictor is applied to the projected values at the half timestep $t^n + \Delta t/2$. The characteristic predictor is formulated in the local system of coordinates for the one-dimensional Euler equation. We illustrate the implementation of

the characteristic predictor in the direction of the unit vector $\mathbf{n}_C$. The Euler equations for this direction can be written in the form

$$W_t + A(W) W_{nc} = 0; \tag{2.13}$$

$$W = \begin{Bmatrix} \tau \\ u \\ p \end{Bmatrix}; \; A(W) = \begin{pmatrix} u & -\tau & 0 \\ 0 & u & \tau \\ 0 & \rho c^2 & u \end{pmatrix}, \tag{2.14}$$

where $\tau = \rho^{-1}$, ρ denotes density, u, p are the velocity and pressure. The matrix $A(W)$ has three eigenvectors ($l\#$, $r\#$) (l for left and r for right, where # stands for +, 0, –) associated with the eigenvalues $\lambda^+ = u + c, \lambda^0 = u, \lambda^- = u - c$.

An approximation of the value projected to an edge, accurate to second order in space and time, can be written as

$$\begin{aligned} W_{i+\Delta r}^{n+1/2} &\approx W_i^n + \frac{\Delta t}{2}\frac{\partial W}{\partial t} + \Delta r \frac{\partial W}{\partial r}_{nc} \\ &\approx W_i^n + \left[\Delta r - \frac{\Delta t}{2} A(W_i)\right] \frac{\partial W}{\partial r}_{nc}. \end{aligned} \tag{2.15}$$

An approximation to $W_{i+\Delta r}^{n+1/2}$ can be written as

$$W_{i+\Delta r}^{n+1/2} = W_i + \left(\Delta \boldsymbol{r}_i - \frac{\Delta t}{2}\left(M_x M_n\right) \boldsymbol{n}_c\right) \cdot \nabla W_i, \tag{2.16}$$

where

$$\left(\mathrm{M}_x \, \mathrm{M}_n\right) = \begin{cases} \max\left(\lambda_i^+, 0\right) & \text{for the cell on the left of the edge} \\ \min\left(\lambda_i^-, 0\right) & \text{for the cell on the right of the edge.} \end{cases} \tag{2.17}$$

The gradients applied in the process of computing the projected values at $t^n + \Delta t/2$ are subjected to the monotonicity limiter.

Following the characteristic predictor described above, the full Riemann problem is solved at the edge. The solution of the Riemann problem defines the flux $Q^{n+1/2}$ through the edge. The fluxes through the edges of triangles are then integrated (Eq. 2.5), thus giving an updated value of the variables at t^{n+1}. One of the advantages of this algorithm is that calculation of the fluxes is done over the largest loop in the system (the loop over edges) and can be vectorized or parallelized. This leads to an efficient algorithm.

We have carried out an extensive and painstaking series of tests in the course of developing and implementing the algorithm. Most of these used a standard benchmark, the exploding diaphragm or "Sod problem" (Fig. 2.1.2). In this problem two regions containing an ideal gas at different densities and pressure are separated by an infinitely thin interface (the diaphragm). A shock wave, a rarefaction wave, and a contact discontinuity propagate away from that point at different speeds when this diaphragm is instantaneously removed. The Riemann solution yields an analytical solution in terms of simple waves which can be compared with the numerical approximation.

We used this problem as a testbed to compare structured vs. unstructured grids, first-order vs. second-order Godunov schemes, schemes with and without limiters, etc. For example, Fig. 2.1.2 shows that the solution obtained with an unstructured grid is noticeably better than that obtained with a structured grid.

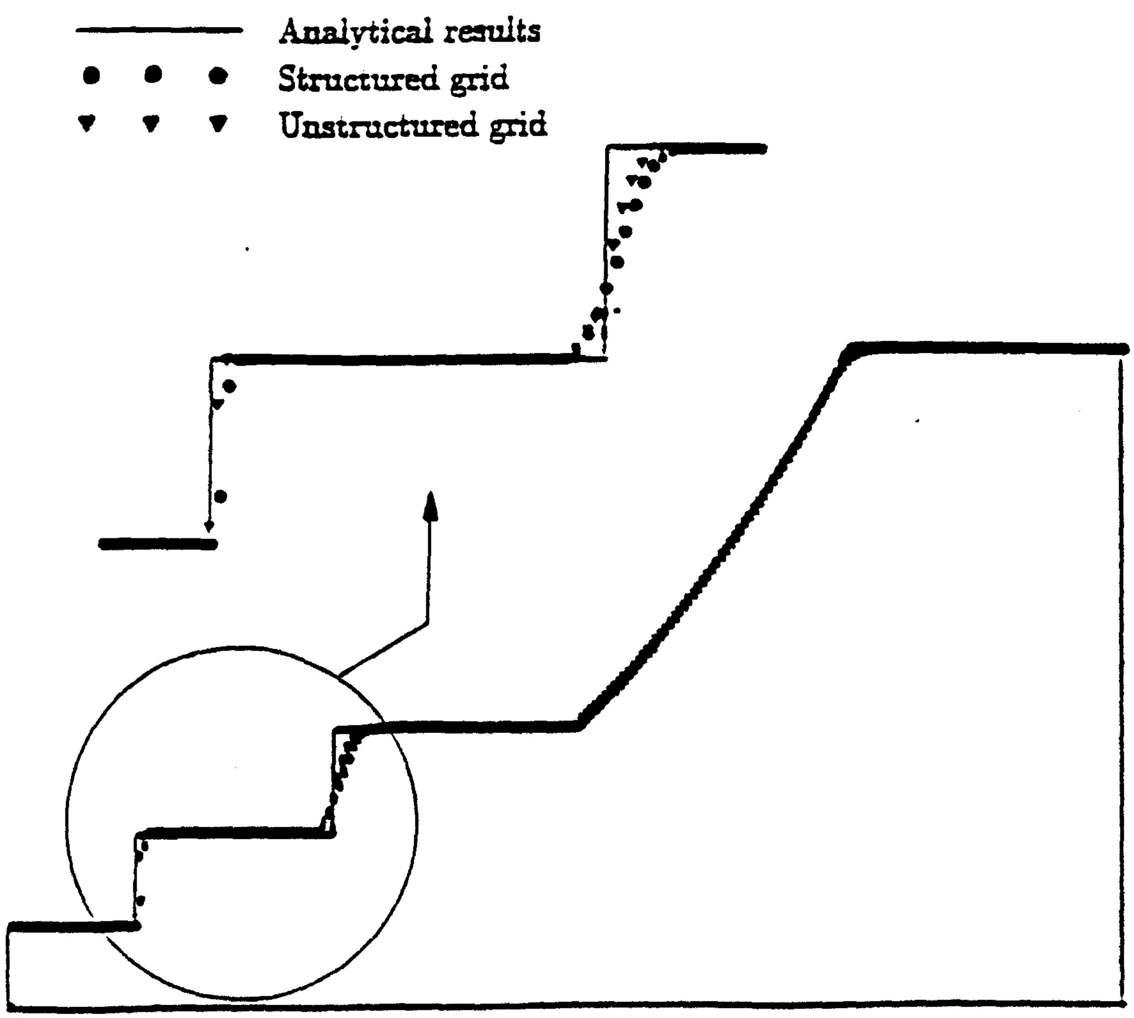


Figure 2.1.2 Density profile comparison between analytical results and results obtained by applying the second-order Godunov algorithm using structured or unstructured grids.

2.2 MULTIPHASE MULTICOMPONENT REACTIVE FLOW

Multiphase multicomponent reacting flows (MPMCRF) consist of material media (continua and particles) dispersed in a flow varying in space and time. Two basic approaches can be used to describe MPMCRFs, heterogeneous and homogeneous phase descriptions. For homogeneous mixtures one assumes that each mixture component occupies the same volume with other mixture components on an equal basis ($V_1=V_2=\ldots=V_n=V$). This approach is justified for an interpenetrating mixture of gases or a dilute suspension of particles in a gas. In a heterogeneous description of a suspension, each component occupies only part of the global volume ($V_1+V_2+\ldots+V_n=V$). Therefore in the mathematical description of the heterogeneous suspensions, in addition to the density of the i-th component ρ_i one needs to introduce the fractional volume of the components:

$$\phi_1 + \phi_2 + ..\phi_N = 1 \quad (\phi_i > 0), \tag{2.18}$$

which allows us to define the real density of each of the components as $\sigma_i = \frac{\rho_i}{\phi_i}$.

Consider a chemically reacting system containing an N-component gaseous phase and one solid particle phase. The conservation equations can be written as follows:[3]

Conservation of Mass

Global continuity for gaseous phase:

$$\frac{\partial \rho_g}{\partial t} + \frac{\partial}{\partial x_j}\left(\rho_g u_{(g)j}\right) = I_g . \tag{2.19}$$

Continuity of N-1 species or components of gaseous phase:

$$\frac{\partial Y^i}{\partial t} + \frac{\partial}{\partial x_j}\left[\rho_g Y^i\left(u_{(g)j} + V^i_j\right)\right] = \omega^i + I^i_g . \tag{2.20}$$

Continuity for solid particle phase:

$$-\frac{\partial \rho_p}{\partial t} + \frac{\partial}{\partial x_j}\left(\rho_p u_{(p)j}\right) = - I_p . \tag{2.21}$$

In the above equation of mass conservation, ρ_g is the partial gas density. The gas volume fraction is ϕ_g. The relation between partial gas density and material density σ_g is $\rho_g = \phi_g \sigma_g$. Similarly, we define the partial phase density ρ_p and material density σ_p. The relation between the two is then $\rho_p = \phi_p \sigma_p$. We assume volume conservation, which is

$$\phi_g + \phi_p = 1. \tag{2.22}$$

The species diffusion velocity V_i^l is calculated through Fick's law:

$$V_i^l = -\frac{D}{Y^l}\frac{\partial Y^l}{\partial x_i}, \tag{2.23}$$

where D is the diffusion coefficient. Finally, we assume mass conservation in all chemical reactions:

$$\sum_{l}^{N} w_i^l = 0 \quad \text{and} \quad I_p = -\sum_{l}^{N} I_g^l = -I_g. \tag{2.24}$$

Conservation of Momentum

Conservation of momentum for the gaseous phase:

$$\begin{aligned}
&\frac{\partial\left(\rho_g u_{(g)i}\right)}{\partial t} + \frac{\partial}{\partial x_j}\left[\rho_g u_{(g)i} u_{(g)j} + \delta_{ij}\phi_g p_g\right] \\
&= \frac{\partial}{\partial x_j}\left[\left(\mu' - \frac{2}{3}\mu\right)\frac{\partial u_{(g)k}}{\partial x_k}\delta_{ij} + \mu\left(\frac{\partial u_{(g)i}}{\partial x_j} + \frac{\partial u_{(g)j}}{\partial x_i}\right)\right] \\
&- F_i^{(p)} + I_p u_{(p)i}.
\end{aligned} \tag{2.25}$$

Conservation of momentum for the particle phase:

$$\frac{\partial\left(\rho_p u_{(p)i}\right)}{\partial t} + \frac{\partial}{\partial x_j}\left[\rho_p u_{(p)i} u_{(p)j} + \delta_{ij}\phi_p p_p\right] = \frac{\partial}{\partial x_j}\left(\tau_{(p)ij}\right) + F_i^{(p)} - I_p u_{(p)i}. \tag{2.26}$$

In the above momentum conservation equations, p_p and p_g are the pressure of the solid particle and gaseous phases respectively, $F_i^{(p)}$ represents the interaction force between the two phases, and $\tau_{(p)ij}$ is the stress tensor for the particle phase, to be determined by experimental or empirical correlations.

For the gaseous phase, the stress tensor can be written as

$$\tau_{(g)ij} = -p\delta_{ij} + \left(\mu' - \frac{2}{3}\mu\right)\frac{\partial u_k}{\partial x_k}\delta_{ij} + \mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right), \tag{2.27}$$

where μ is the dynamic viscosity and μ' is the second viscosity coefficient.

Conservation of Energy

The governing equation for conservation of energy for the gaseous phase is usually written

$$\frac{\partial\left[\rho_g\left(e_g+0.5u_{(g)i}u_{(g)i}\right)\right]}{\partial t}+\frac{\partial}{\partial x_j}\left[\rho_g u_{(g)j}\left(e_g+0.5u_{(g)i}u_{(g)i}\right)+\phi_g p_g u_{(g)j}\right]$$

$$=-\frac{\partial q_{(g)j}}{\partial x_j}+Q_g+\frac{\partial}{\partial x_j}\left(u_{(g)i}\left[\left(\mu'-\frac{2}{3}\mu\right)\frac{\partial u_{(g)k}}{\partial x_k}\delta_{ij}+\mu\left(\frac{\partial u_{(g)i}}{\partial x_i}\right)\right]\right) \tag{2.28}$$

$$-F_{(p)i}u_{(p)i}+Q_p\,.$$

The equation for conservation of energy for the particle phase has the form

$$\frac{\partial}{\partial t}\left[\rho_p\left(C_s T_p+0.5u_{(p)i}\right)\right]+\frac{\partial}{\partial x_j}\left[\rho_p u_{(p)j}\left(C_s T_p+0.5u_{(p)i}u_{(p)i}\right)+\phi_p u_{(p)j}p_p\right]$$

$$=-\frac{\partial q_{(p)j}}{\partial x_j}+\frac{\partial}{\partial x_j}\left(u_{(p)j}\tau_{(p)ij}\right)+F_i^{(p)}u_{(p)i}-Q_p\,. \tag{2.29}$$

In the conservation of energy, $\frac{\partial q_{(g)j}}{\partial x_j}$ and $\frac{\partial q_{(p)j}}{\partial x_j}$ are the heat flux gradients in the *j*th direction in the gaseous and particle phases, respectively. Q_p is the energy source due to heterogeneous chemical reactions (between the gaseous and particle phases), plus heat transfer between the two phases. Here $Q_g=\sum_{i=1}^{N}\left(-\omega_i\Delta h_{fi}^{o}\right)$ is the energy source due to homogeneous (gaseous) chemical reactions, which is defined in the chemical reaction model.

Conservation of Number of Particles

An equation for total conservation of particles is given by

$$\frac{\partial n_p}{\partial t}+\frac{\partial}{\partial x_j}\left(n_p u_{(p)j}\right)=0\,. \tag{2.30}$$

Equation of State

The equation of state for all gases can be put into the generic form

$$e_g=f_g\left(p_g,\sigma_g,Y^1,\cdots,Y^N\right), \tag{2.31}$$

where for an ideal gas the form is

$$e_g = \frac{p_g}{\sigma_g(\gamma_g - 1)} \tag{2.32}$$

and
$$p_g = \sigma_g R_u T_g \sum_{i=1}^{N} \frac{Y^i}{W^i}. \tag{2.33}$$

An equation of state for the particle phase can be written in symbolic form as

$$p_p = f(\sigma_p, T_p), \tag{2.34}$$

where the exact form of Eq. (2.34) that is to be used in a numerical simulation depends on experimental data or results from physical approximations.

In the above equations, γ_g is the ratio of specific heats of the gaseous mixture and R_u is the universal gas constant.

Chemical Reaction Model

A phenomenological chemical reaction model for the gaseous phase (including M chemical reactions) has been formulated as

$$\omega^i = W^i \sum_{k=i}^{M} \left(\nu_k'^{(1)} - \nu_k^{(1)}\right) B_k T^{\alpha k} \exp\left(\frac{E_{\alpha k}}{R_u T_g}\right) \prod_{j=1}^{N} \left(\frac{X^j p_g}{R_u T_g}\right)^{\nu_{jk}}. \tag{2.35}$$

Similarly, a phenomenological heterogeneous (for gas and particle phases) chemicals reaction model can be written symbolically as

$$I_{(p)} = f(T_p, P_p, \ldots), \tag{2.36}$$

and again the exact form of Eq. (2.36) will depend on experimental data or approximations from physical models.

The following nomenclature defines the symbols used in the above system of equations (2.19) - (2.36): B – chemical reaction collision frequency factor; C_S – specific heat for solid particle; e – internal energy; D – mass diffusion coefficient; $E_{\alpha k}$ – activation energy for the kth reaction; F_i – interphase force in ith direction; I – source function generated by chemical reaction; p_g – gas pressure; q_i – heat flux in the ith direction; R_u – universal gas constant; t – time; T – temperature; u_i – velocity in ith direction; V_i – species diffusion velocity in ith direction; W^i – molecular weight of ith component of gas; x_i – coordinate in ith direction; X^i – mode fraction of ith component of gas; Y^i – mass fraction of ith component of gas; α – temperature exponent of the kth reaction; γ – ratio of specific heat; λ – thermal conductivity of gas; μ – dynamic viscosity of gas; μ' – second

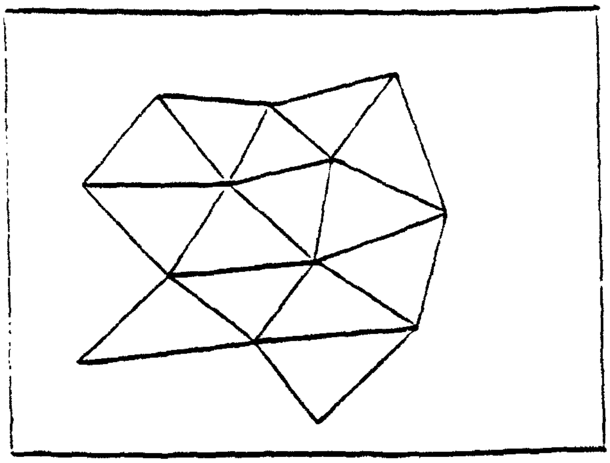
viscosity coefficient of gas; τ_{ij} – stress tensor; ω^{i} – mass rate of production of species i; ρ – density; $\nu_{i,k}$ – stoichiometric coefficient for species i appearing as a reactant in the kth reaction; $\nu'_{i,k}$ – stoichiometric coefficient for species i appearing as a product in the kth reaction; ϕ – volume fraction; σ – material density. Subscripts are defined as follows: g – gas phase; p – particle phase; i,j,k, – direction indexes; l – species index. Superscripts refer to species type.

The comprehensive mathematical model and system of equations given above for an MPMCRF simulation of advanced material synthesis processes is based on volume averaging, assuming that each phase or component can be described by continuous flow. Such averaging leads to a loss of information that can be recovered by appropriate closure relations. The closure relations such as interphase forces, chemical reaction models and the equations of state are usually developed from correlations involving experimental data or from simple physical or chemical models describing interphase or intraphase interactions. Such correlations are generally only valid within the range of known experimental data; the choice of appropriate closure models reflects the understanding of the underlying physical and chemical nature of the system to be simulated.

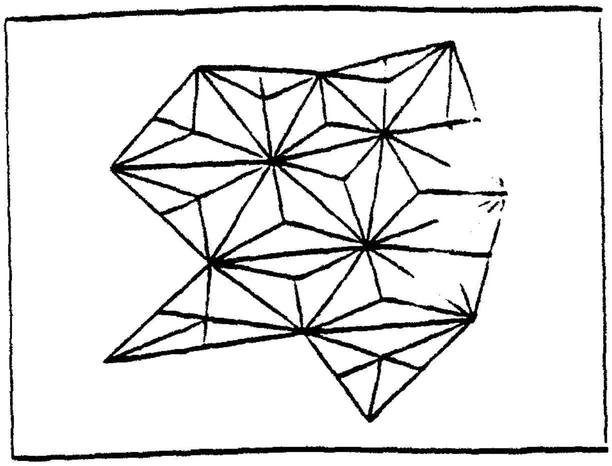
2.3 DIRECT DYNAMIC REFINEMENT METHOD FOR UNSTRUCTURED TRIANGULAR GRIDS

As stated, an unstructured grid is very well suited to implement boundary conditions on complex geometrical shapes and to refine the grid if necessary. This feature of the unstructured triangular grid is compatible with efficient use of memory resources. The adaptive grid enables the code to capture moving shocks and large-gradient flow features with high resolution. The memory resources available can be very efficiently distributed in the computational domain to accommodate the resolution needed to capture the main features of the solution's physical property. Dynamic refinement controls the resolution priorities. These priorities can be set according to the physical features that the user wishes to emphasize in the simulation. The user has control over the resolution of the physical features, without being restricted to the initial grid. The alternative to Direct Dynamic Refinement is the hierarchical dynamic refinement[6] (H refinement) that keeps a history of the initial grid (mother grid) and the subdivision of each level (daughter grids). In the H refinement method, it is necessary to keep overhead information on the level of each triangle subdivision, and double indirect indexing is required to keep track of the H refinement process. As mentioned, H refinement relies heavily on the initial grid as it subdivides the mother grid, and returns to that grid after the passage of the shock.

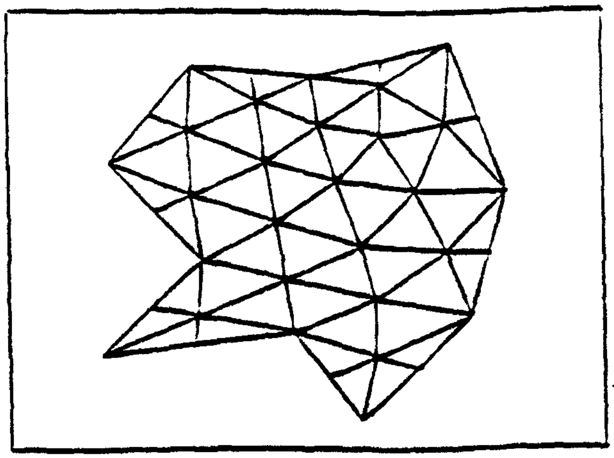
The Direct Dynamic Refinement (DDR) method for capturing the shock requires the refinement to be in the region ahead of the shock. This requirement minimizes the dissipation in the interpolation process when assigning values to the new triangles created in the refined region. Additionally, it requires that the coarsening of the grid be done after the passage of the shock. The interpolation and extrapolation in the refinement and coarsening of the grid is done in the region where the flow features are smooth.



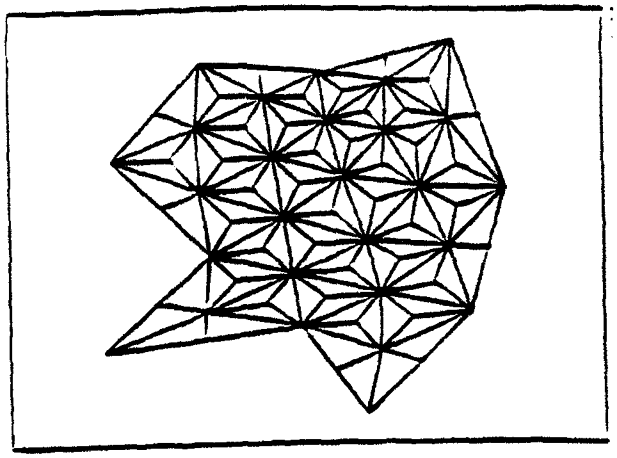
a. Original grid.



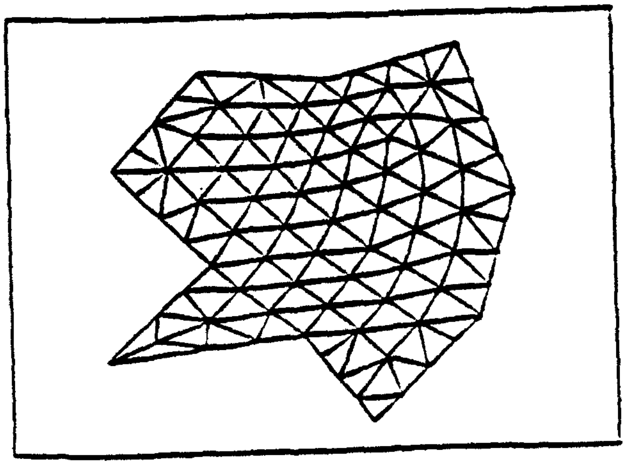
b. Grid after one refinement.



c. Grid after one refinement
and one reconnection.

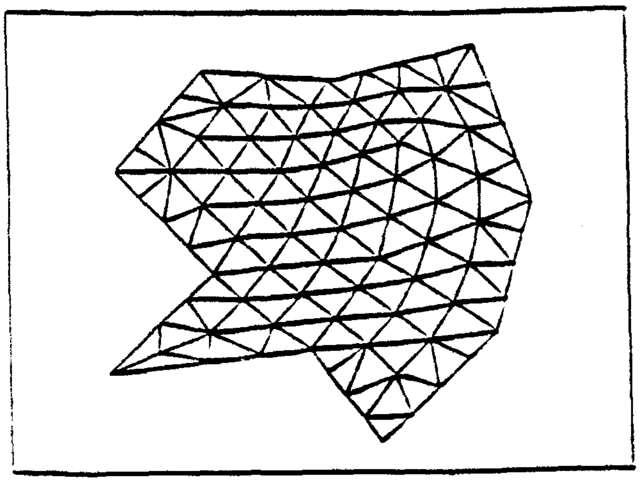


d. Second refinement.

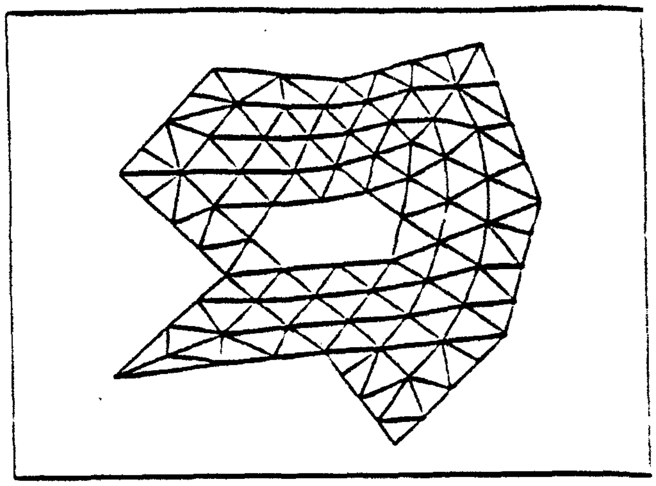


e. Second reconnection.

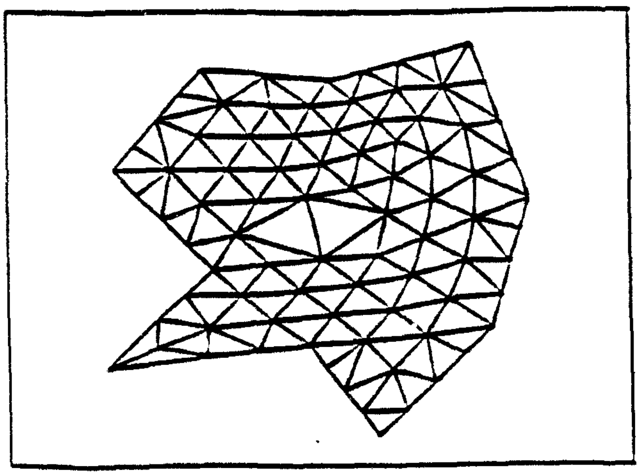
Figure 2.3.1 Illustration of the grid refinement process.



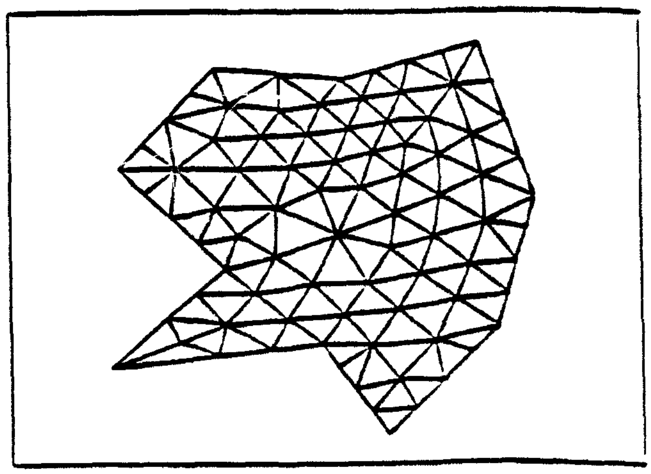
a. Original grid.



b. Point removal.



c. Constructing of new cells.



d. Grid after reconnection and relaxation.

Figure 2.3.2 Illustration of the grid coarsening process.

The physics of the problem is involved in the process that identifies the region of refinement and coarsening. Error criteria can be derived that will allow grid adaptation to stationary or moving pressure or density discontinuities, region of high vortical activity, etc. There should be an error indicator specially suited to capture and identify the region of importance for each of the physics features to be resolved.

The original FUGGS algorithm reported in Ref. 9 was modified to enable adaptivity of the grid in the course of the computation. In AUGUST, we have implemented an algorithm with multiple criteria for capturing a variety of features that might exist in the physics of the problem to be solved. To identify the location of a moving shock, we use the flux of total energy into triangles. The fluxes entering and leaving triangles are the most accurate physical variables computed by the Godunov algorithm for solving the Euler equations, and are used to update the physical variables for each timestep in each triangle. A shock wave means that there is a "step-function" change in the cell that is caused by an influx of energy, momentum or density. Stationary shocks can be identified by density gradients that are computed in the course of implementing the Second-Order Godunov algorithm.

In Fig. 2.3.1, we illustrate the basic process of refinement accomplished in the DDR. The original grid is shown in Fig. 2.3.1a. Figure 2.3.1b illustrates a one-step scheme refinement in which a new vertex is introduced into a triangular cell, forming three new cells. This is followed by reconnection, which modifies the grid as demonstrated in Fig. 2.3.1c. The process of refinement and reconnection can be continued until the necessary grid resolution is achieved, as illustrated in Figs. 2.3.1d and 2.3.1e. This direct approach to the grid refinement provides extreme flexibility in resolving local flow features. A similar simple method is applied to grid coarsening. In the first step of coarsening the marked vertices, all associated elements of the grid are simply removed, as shown in Fig. 2.3.2a. During the second step, this void in the grid is filled with new larger triangles (Fig. 2.3.2b) and then reconnected as shown in Fig. 2.3.2c. When a very large increase of the local grid density is required, these simple algorithms of grid addition and deletion can create triangles with an unacceptably large aspect ratio. To avoid this condition for very large grid densities (when the area of the triangles in the dense region is reduced to less than 2% of the initial area), we introduced local grid relaxation immediately after the grid deletion procedure.

AUGUST has proven to be a very robust and efficient algorithm capable of computing transient phenomena, and with the ability to sense the region of physical interest and resolve it by refining and coarsening the grid as needed.

2.4 STRUCTURED/UNSTRUCTURED COMPOSITE GRIDS

Structured rectangular grids allow the construction of numerical algorithms that perform an efficient and accurate integration of fluid conservation equations. The efficiency of these schemes results from the extremely low storage overhead needed for domain decomposition and the efficient and compact indexing that also defines domain

connectivity. These two factors allow code construction based on a structured domain decomposition that can be highly vectorized and parallelized. Integration in physical space on orthogonal and uniform grids produces the highest possible accuracy of the numerical algorithms. The disadvantage of structured rectangular grids is that they cannot be used for decomposition of computational domains with complex geometries.

The early developers of computational methods realized that, for many important applications of Computational Fluid Dynamics (CFD), it is unacceptable to describe curved boundaries of the computational domain using the stair-step approximation available with the rectangular domain decomposition technique. To overcome this difficulty, the techniques of boundary-fitted coordinates were developed. With these techniques, the computational domain is decomposed into quadrilaterals that can be fitted to the curved domain boundaries. The solution is then obtained in the physical space using the geometrical information defining the quadrilaterals, or in the computational coordinate system that is obtained by transformation of the original domain into a rectangular domain. The advantage of this technique is that it employs the same indexing method as the rectangular structured domain decomposition methods that also serve to define domain connectivity. The boundary-fitted coordinate approach leads to efficient codes, with approximately a 4:1 penalty in terms of memory requirement per cell as compared with rectangular domain decomposition. However, this approach is somewhat restricted in its domain decomposition capability, since distortion or large size variations of the quadrilaterals in one region of the domain leads to unwanted distortions or increased resolution in other parts of the domain. An example of this is the case of structured body fitted coordinates used for simulations of flows over a profile with sharp trailing edges. In this case, increased resolution in the vicinity of the trailing edge leads to increased resolution in the whole row of elements connected to the trailing edge elements.

The most effective methods of domain decomposition developed to overcome this disadvantage are those using unstructured triangular grids. These methods were developed to cope with very complex computational domains. The unstructured grid method, while efficient and powerful in domain decomposition, results in codes that must store large quantities of information defining the grid geometry and connectivity, and have large computational and storage overheads. As a rule, an unstructured grid code requires greater storage by a factor of 10, and will run about 20 times slower per cell per iteration than a structured rectangular code.

Unstructured grid methods are used to their best advantage when combined with grid adaptivity. This feature usually allows dynamic decomposition of the computational domain subregions, thus leading to an order-of-magnitude reduction in the number of cells for some problems, as compared to the unstructured grid lacking this adaptive capability. However, this advantage is highly dependent on the problem solved. Adaptive unstructured grids have an advantage over the unadaptive unstructured domain decomposition if the area of high-resolution domain decomposition is less than one tenth of the global area of the computational domain. This explains why the adaptive unstructured method may be extremely effective for solutions with multiple shock waves

in complex geometries, but becomes extremely inefficient when high resolution is needed in a substantial area of the computational domain.

Our approach to domain decomposition combines the structured and unstructured methods for achieving better efficiency and accuracy. Under this method, structured rectangular grids are used to cover most of the computational domain, and unstructured triangular grids are used only to patch between the rectangular grids (Fig. 2.4.1) or to conform to the curved boundaries of the computational domain (Fig. 2.4.2). In these figures, an unstructured triangular grid is used to decompose the regions of the computational domain that have a simple geometry.

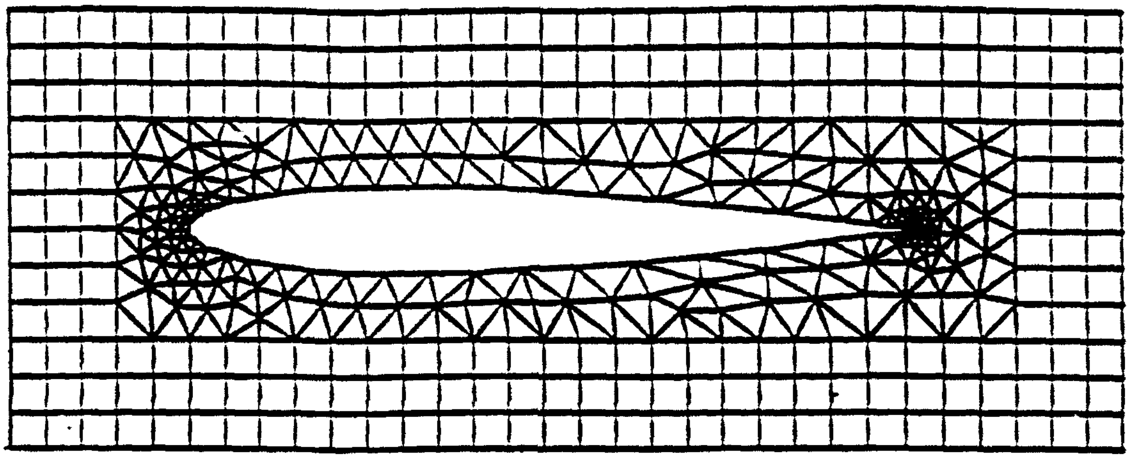


Figure 2.4.1 A possible candidate configuration for hybrid structured/unstructured domain decomposition.

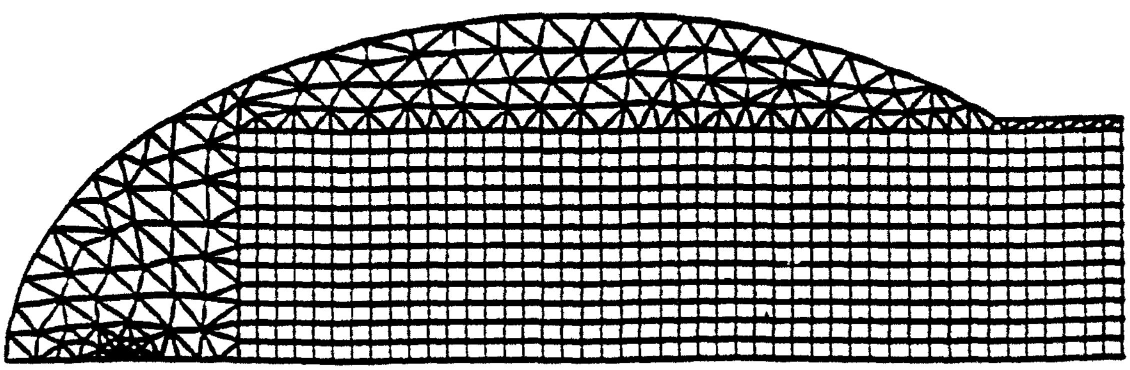


Figure 2.4.2 Hybrid structured/unstructured grid used to simulate ellipsoidal reflector, showing adaptation to curved boundaries.

2.5 THREE-DIMENSIONAL CAPABILITY

Once the 2D capability was fully developed, we initiated the development of a fully 3D CFD adaptive unstructured simulation capability. This part of our effort is not yet documented in published material.

The first step in solving a 3D CFD problem is to discretize the computational domain into tetrahedra. The grid generation is a recognized bottleneck in the time it takes to evaluate an aerodynamic configuration.[15] One could even argue that it represents the most time-consuming portion of the evaluation process. There are a handful of codes that are capable of gridding any given domain into tetrahedra. In order to shorten the part to our objective of achieving a 3D adaptive solver capability, we decided to make use of an existing grid generator to provide the initial grid.

OCTREE,[16] which was developed at Rensselaer Polytechnic Institute (RPI), is a Finite Octree 3D grid generator that provides the initial grid for our adaptive solver. The productivity of a 3D grid generator is a function of the complexity of the surfaces that define the domain of computation. Usually, this task is the most time-consuming and painful for the user. OCTREE does not have a CAD/CAM package to assist the user in defining the surfaces of the geometry to be gridded. Nevertheless, OCTREE is a very robust and reliable grid generator.

The OCTREE algorithm is based on the concept of dividing the computational domain into octants. In each step, the code defines three planes that halve the domain in each of the three dimensions, thus dividing the volume into eight octants. Those three planes intersect the surfaces of the geometry, defining vertices. All the vertices are collected and sorted into topological loops. If the vertices are not sufficient to define correct topological loops, the code will subdivide the corresponding octant into eight smaller octants until the topology is fully resolved. The user is allowed to specify the level of the local octree subdivision he wishes to resolve. Once the code subdivides the volume into the level of octree specified by the user or needed to resolve the local geometrical details, the code defines tetrahedra to fill the volume of the computation domain. The code provides the user with an option that improves the quality of the tetrahedra by smoothing and eliminating the very small ones.

As stated, OCTREE provides the initial grid for the 3D solver. The adaptivity of the mesh is controlled by specific physical features that the user defines based on the physics of the problem to be solved. The adaptivity of the mesh automatically traces the physical features in the simulation and refines and coarsens the mesh accordingly to the criteria and the resolution specified by the user.

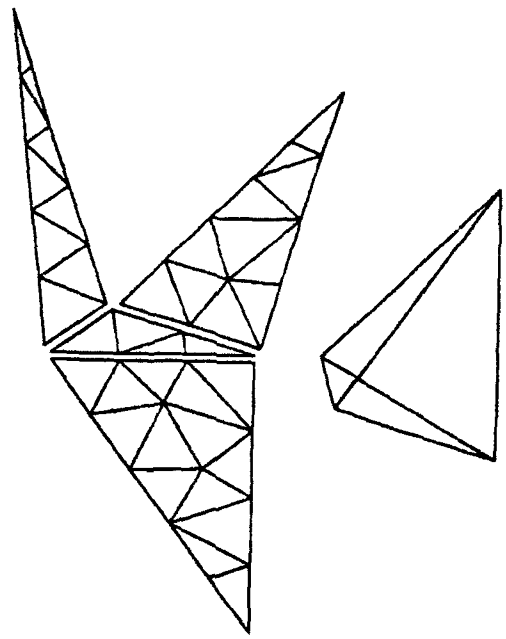


Figure 2.5.1 An elongated tetrahedron can be refined using smaller tetrahedra that are nearly regular.

The target tetrahedra are refined by first subdividing each of the four surfaces into smaller triangles that satisfy the resolution set by the user. There are no constraints on the way each face is subdivided. Each edge of the face is subdivided according to the local resolution needed, and the points along the edges are connected to construct the best triangles possible. The code adds points inside the face along with points on the edges to achieve an adequate triangulation of the faces. The triangles of the four faces of the target tetraheda are used to define smaller tetrahedra that will fill the volume. If needed, the code will add points inside the volume of the target tetrahedron to achieve the best tetrahedra possible. The code has the ability to reconnect tetrahedra to improve quality. The reconnection is done by pulling out an edge, sorting all the tetrahedra connected to this edge, deleting these tetrahedra and filling the void with better shaped tetrahedra.

Figure 2.5.1 shows how the subdivision process can fill an irregular (elongated) tetrahedron with smaller tetrahedra that are nearly equilateral. (This is not the case with H refinement.) Figure 2.5.2 shows points used to create octree refinement to grid a problem involving surface-mine blast effects on the underside of a truck. Figure 2.5.3 is the corresponding tetrahedral grid. The calculated overpressures on the surface of the truck underbody for an eight-pound explosive are shown in Sec. 3.4.

The algorithm used to solve the 3D gasdynamic equations is an immediate extension of the 2D case described in Sec. 2.1. Thus, Eq. (2.6) is replaced by

$$\int_{\Omega} \nabla U_i^{\text{cell}} \, \mathrm{d}V = \int_{\partial\Omega} U_j^{\text{face}} \, \mathrm{d}\mathbf{S}, \tag{2.6'}$$

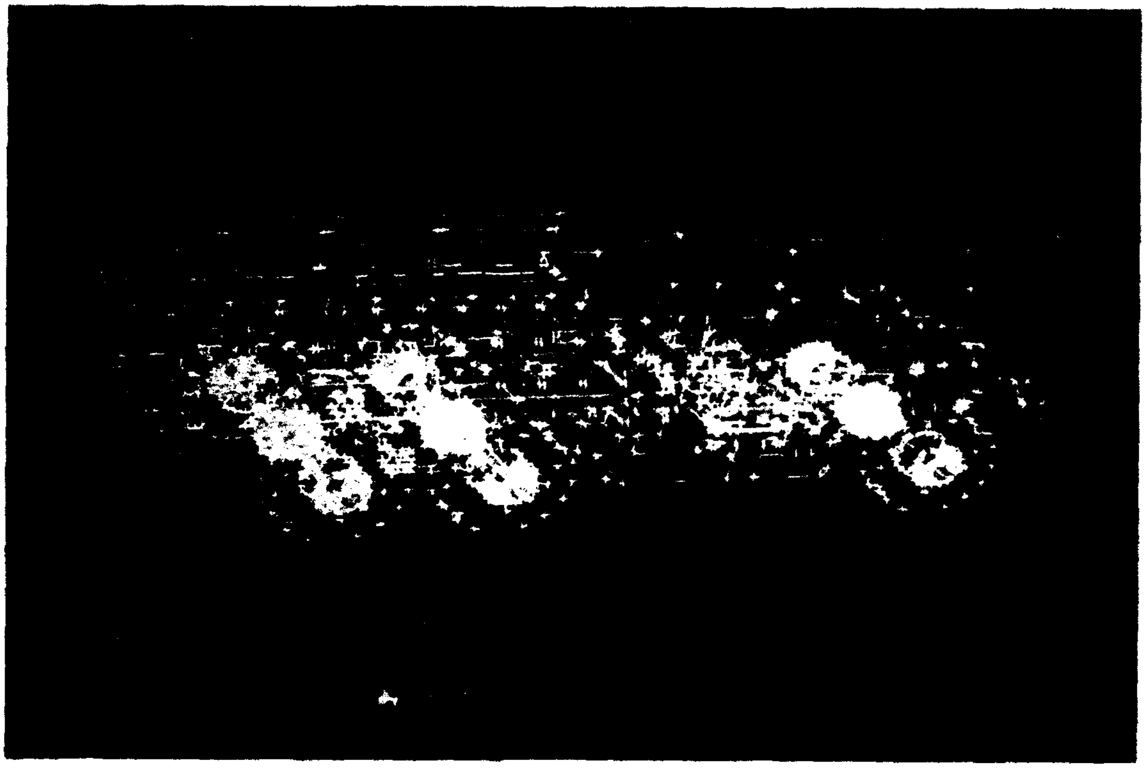


Figure 2.5.2 Points used to define structure of vehicle.

Figure 2.5.3 Tetrahedral grid generated by Finite Octree method.

where now Ω and $\partial\Omega$ are the volume and surface of a tetrahedron, and dV and dS are the corresponding differential elements. Its finite-difference approximation is

$$\nabla U_i^{cell} = \frac{1}{V}\sum_{j=1}^{4} \tilde{U}_j^{face}\, \mathbf{n}_j\, \Delta S_j, \tag{2.7'}$$

where the summation is over the four faces and $\mathbf{n}_j$ is the normal to the jth face with surface area dS_j. In the equations corresponding to Eqs. (2.9) - (2.11), the range 1, 2, 3 is replaced by 1, 2, 3, 4. Equations (2.12) - (2.16) are formally unchanged, and Eq. (2.17) becomes

$$\left(M_x\, M_n\right) = \begin{cases} \max\left(\lambda_i^{+}, 0\right) & \text{for the cell on the left of the edge} \\ \min\left(\lambda_i^{-}, 0\right) & \text{for the cell on the right of the edge.} \end{cases} \tag{2.17'}$$

3. APPLICATIONS

The AUGUST code was extensively validated for a wide range of known CFD problems and has been shown to be a robust simulation tool. It has been utilized on a variety of problems which span flow regimes ranging from low subsonic Mach numbers to hypersonic Mach numbers (Table 3.1).

Appendix C contains a complete collection and description of the CFD problems addressed during the UUGM research. Additional details of the AUGUST code are contained in SAIC's progress report for the UUGM DARPA program, submitted in November 1991. Here we briefly describe the most noteworthy applications.

It is worth underscoring again that in the past it was necessary to use a sequence of codes as well as numerical parameter adjustment to bridge the gap in flow phenomena occurring in different flow regimes. An important point to be made here is that the AUGUST code allows robust, accurate and efficient solutions across these different regimes without the necessity of adjusting coefficients to enhance convergence accuracy or efficiency.

Table 3.1 AUGUST Applications

Problem	Activity
1. Calculation of potential flow about an ellipse.	Reported at the 4th International Symposium on Computational Fluid Dynamics, Davis, CA, Sept 1991.
2. Hypersonic flow past a flat plate.	Reported at AIAA Reno Meeting (AIAA-90-0699), 1990.

Problem	Activity
3. Shock on wedge with adaptive gridding.	Reported at the Free Lagrange Conference, Jackson Lake, WY, 1990.
4. Simulation of mine explosion under a vehicle.	Performed for U.S. Army Corps of Engineers, Ft. Belvoir, VA.
5. Simulation of pulsed detonation engine.	Published in J. of Propulsion and Power Nov/Dec 1991 Vol. 7 (6) pp. 857-865 and AIAA Meeting, Reno, NV 1992.
6. Shock focusing in air using structured/unstructured grids.	Presented at the ICAM Conference, Rutgers, NJ, June 1992.
7. Nonideal airburst calculations for multiphase media.	Performed for the Defense Nuclear Agency, Alexandria, VA.
8. Flow in the SARL wind tunnel.	Performed for Wright-Patterson AFB.
9. Simulation of a shock on a double wedge.	Presented at the Army workshop on Adaptive Methods for PDEs, RPI, March 1992.
10. Supersonic spray coating devices.	To be published.
11. Nanomaterial synthesis.	Published in Surf. Coating Tech. **49**, 387-393 (1991).
12. Dusty flow over a cylinder.	To be published in AIAA Journal.
13. Image processing.	Presented at SPIE conference on Applications of Digital Image Processing, San Diego, July 1991.
14. Multiphase detonation.	Published in Combust. Sci. Tech. **89**, 201-218 (1993).

3.1 POTENTIAL FLOW OVER AN ELLIPSE

One of the outstanding early CFD computational challenges (from the point of view that no satisfactory solution had been obtained) was associated with simulating subsonic (Mach 0.2 and less) flow over a symmetric elliptical airfoil using the Euler equations (Fig. 3.1.1). All previous attempts to compute the flow over such an ellipse resulted in spurious lift and drag values that were significantly larger than the classical

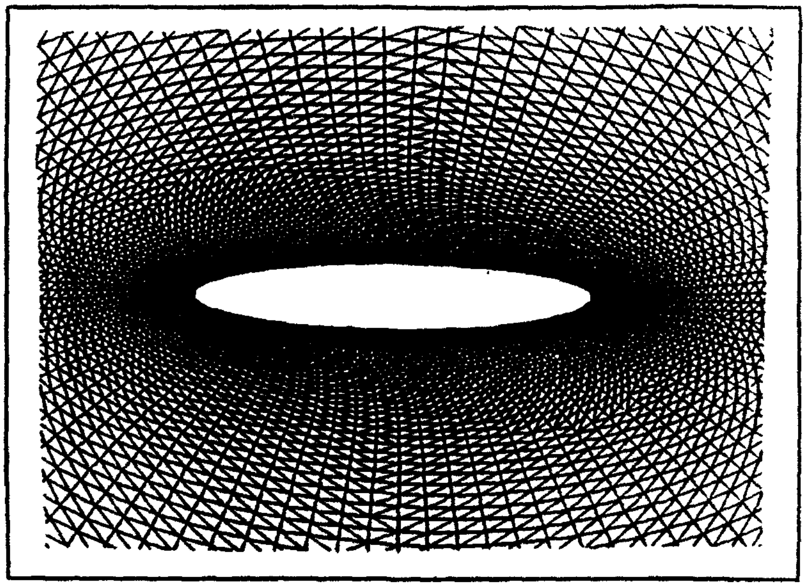


Figure 3.1.1 The grid used for simulating the flow over an ellipse.

potential flow solution. The potential flow result should have been closely approximated if there were no numerical viscosity present. This test case is important because, in transitioning from an Euler solver to a full Navier-Stokes solver, one needs confidence that the artificial (numerical) viscosity will not dominate the physical viscosity included in the Reynolds' stress terms. As shown in Appendix C-1, use of an earlier version of the AUGUST code, the Fast Unstructured Grid Godunov Solver (FUGGS) code provided solutions to this test case that were very close to the potential flow solution. Other attempts resulted in lift and drag values that were off by several orders of magnitude compared with the SAIC FUGGS results. The results described here were prepared for a poster presented to Dr. Arje Nachman, SAIC's UUGM AFOSR program monitor and Dr. James Crowley, SAIC's UUGM DARPA program manager.

3.2 HYPERSONIC FLOW PAST A FLAT PLATE

To demonstrate the versatility of the method for the entire range of flow regimes we have simulated a hypersonic flow test problem. One of the advantages of the Godunov methods is that over the whole range of calculations performed (low subsonic flow, supersonic flow, unsteady flow with strong shock, or hypersonic flow at Mach number M=32) it is unnecessary to change or adjust the numerical algorithm. In Ref. 17 the performance of first- and second-order Godunov methods was analyzed for hypersonic flow regimes. There, as a test problem, an analytical solution was used for a hypersonic flow around a flat plate of finite thickness. This solution was obtained based on the analogy between hypersonic flow over a flat plate of finite thickness and a strong planar explosion. Here we use an expression from Ref. 17 which defines the shape of the shock wave as a function of plate thickness d; γ is the adiabatic coefficient, and α is a nondimensional scale factor related to the energy released at the stagnation point.

$$Y_{\text{SHOCK}} = \left(\frac{1}{2} D_f \frac{dx^2}{2} \right)^{1/3}$$

where D_f is a coefficient of order unity,

$$a = k_1 (\gamma - 1)^{k_2 + k_3 \ln(\gamma - 1)}$$

with $k_1 = 0.36011$, $k_2 = 1.2537$, and $k_3 = -0.1847$.

As a direct comparison we solved the hypersonic flow problem for the same set of conditions as in Ref. 17:

$$U_\infty = 10011 \text{ m/sec}, \; p = 98.72 \text{ Pa}, \; \rho = 1.24 \times 10^{-3} \text{ kg/m}^3, \text{ and } \gamma = 1.2.$$

The grid used for this simulation is shown in Fig. 3.2.1a. This grid has ≈ 5500 vertices and its spatial resolution at the leading edge of the plate is of the same order as that of a 300 x 60 rectangular grid used in Ref. 5.

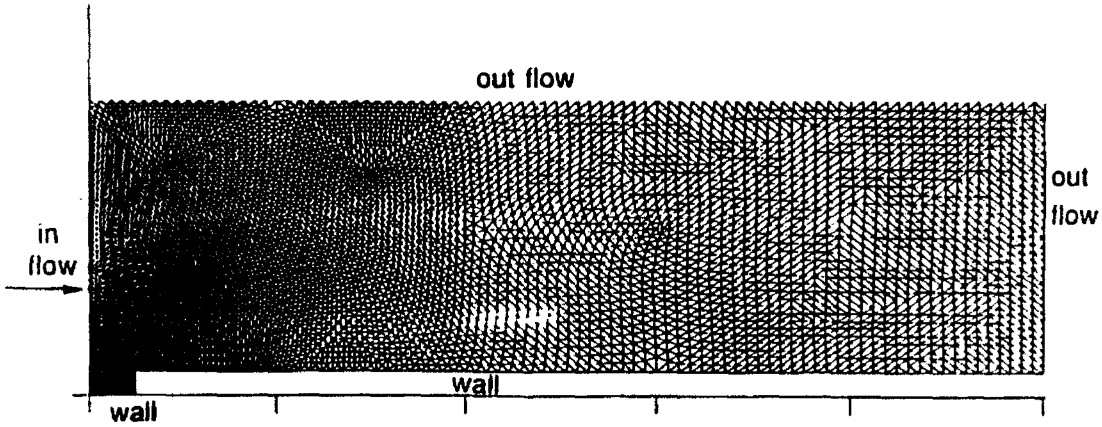


Figure 3.2.1a Grid for simulation of hypersonic flow over a flat plate.

Figure 3.2.1b shows results for this simulation in the form of pressure contours. Figure 3.2.1b also represents the location of the analytically calculated shock front by a discrete line (squares). The shock resolution and accuracy or its location are comparable to that obtained in Ref. 17 even though our triangular grid has less than one third as many nodes as the rectangular grid used in Ref. 17. This is because in constructing the triangular grid we had the flexibility to place the highest concentration of nodes in the area of the leading edge where the main properties of the flow are established.

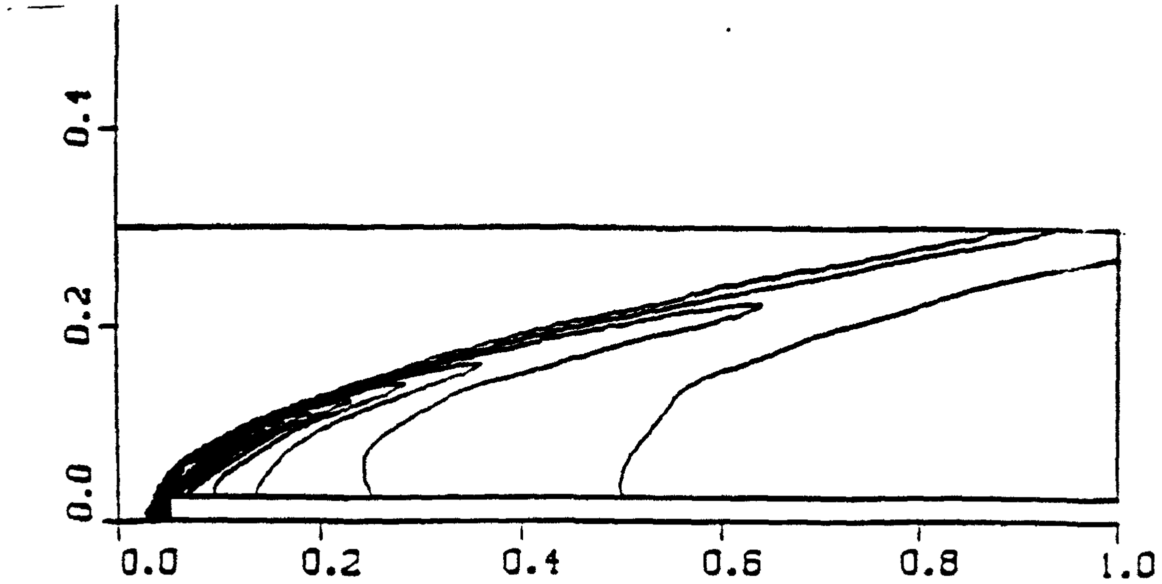


Figure 3.2.1b Second order solution for a flat plate, pressure contours. Mach = 32: 5509 grid vertices: $P_{max} = 5.0 \times 10^4$ Pa, $P_{min} = 98.7$ Pa.

3.3 SHOCK ON WEDGE WITH ADAPTIVE GRIDDING

An unstructured grid is very suitable for implementing boundary conditions on complex geometrical shapes and refining the grid if necessary. This feature of the unstructured triangular grid is compatible with efficient usage of memory resources. The adaptive grid enables the code to capture moving shocks and large-gradient flow features with high resolution. The memory resources available can be very efficiently distributed in the computational domain to accommodate the resolution needed to capture the main features of the physical property of the solution.

One strategy for doing this is called hierarchical dynamic refinement (H refinement). It keeps a history of the initial grid (other grid) and the subdivision of each level (daughter grid). H refinement subdivides the initial grid into two or four triangles in each level, and keeps track of the number of subdivision levels each triangle has undertaken. In the H refinement method, one has to keep overhead information on the level of each triangle subdivision, and needs double indirect indexing to keep track of the H refinement process. This slows down the computation by partially disabling the vectorization of the code. As mentioned, H refinement relies heavily on the initial grid as it subdivides the mother grid and returns back to it after the passage of the shock.

AUGUST and its predecessor FUGGS use a second-order Godunov solver on an unstructured grid. The refinement strategy incorporated in these codes is called Direct Dynamic Refinement. For shock capturing, Direct Dynamic Refinement basically requires

the refinement to be in the region ahead of the shock. This requirement minimizes the dissipation in the interpolation process when assigning values to the new triangles created in the refined region. Additionally, it requires that the coarsening of the grid should be done after the passage of the shock. In principle, the interpolation and extrapolation in the refinement and coarsening of the grid are done in the region where the flow features are smooth.

FUGGS was used with direct dynamic refinement to solve the transient behavior of the flow entering a channel with a wedge (prism) having an inclination of 27°. The flow enters the channel from the left with Mach number 8.7. A sequence of snapshots illustrates the density contours, and the grid for each timestep is given in Figs. 3.3.1a - 3.3.3a (contour plots) and 3.3.1b - 3.3.3b (grid). These figures clearly demonstrate the automatic adaptation of the grid to the moving shocks and the ability to capture the detailed physics of the simulation with very high resolution and minimal memory requirements. The initial grid can clearly be seen to the right of the shock ("ahead") in the early stage of the shock propagation from left to right. The coarsening algorithm is able to produce a reasonable mesh in the region trailing the shock as shown in the figures.

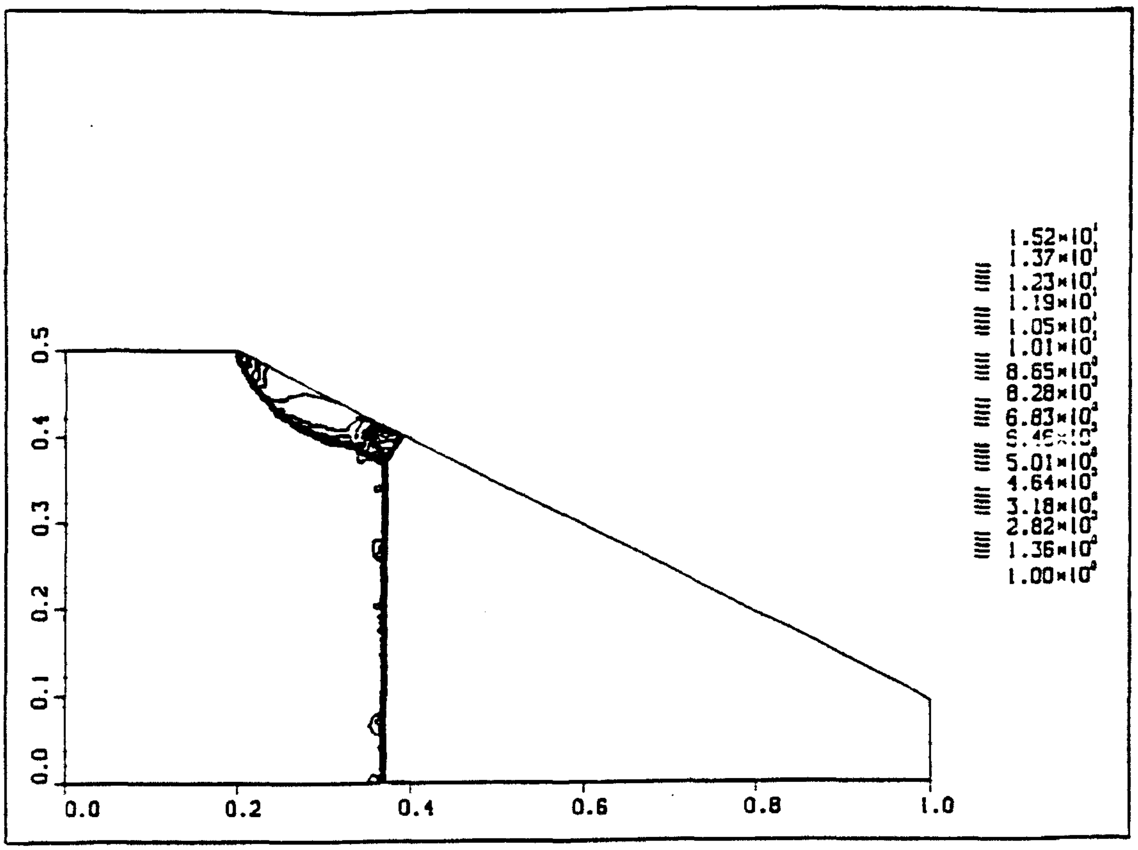


Figure 3.3.1a Density contours at early time for shock in planar channel (M = 8.7, wedge angle = 27°).

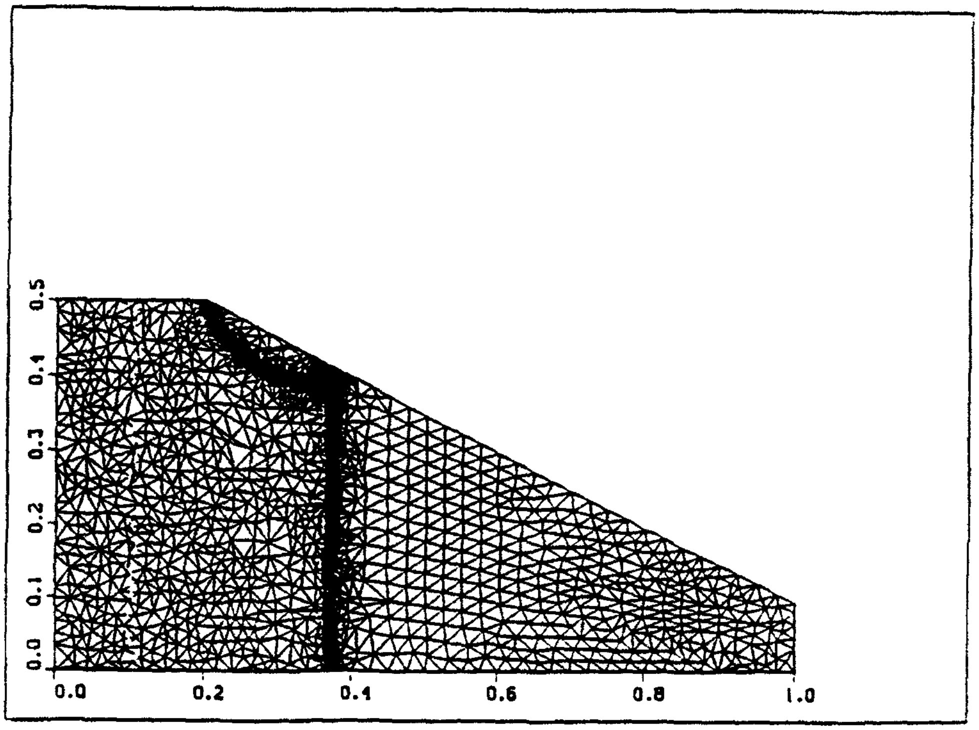


Figure 3.3.1b Grid at early time for shock in planar channel
(M = 8.7, wedge angle = 27°).

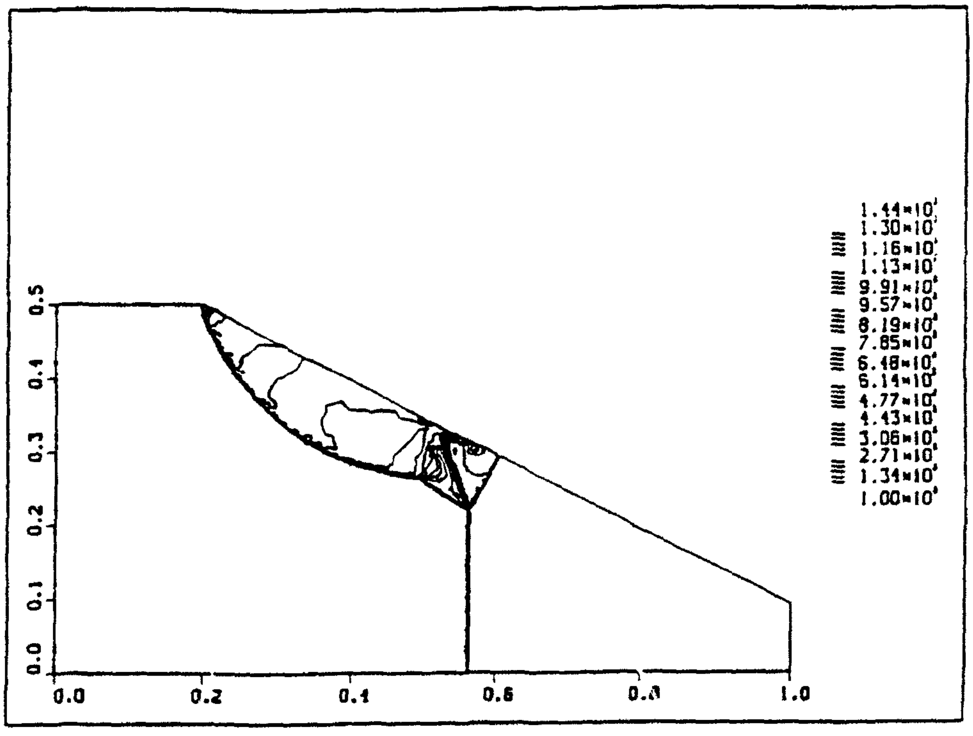


Figure 3.3.2a Density contours at intermediate time for shock in planar channel
(M = 8.7, wedge angle = 27°).

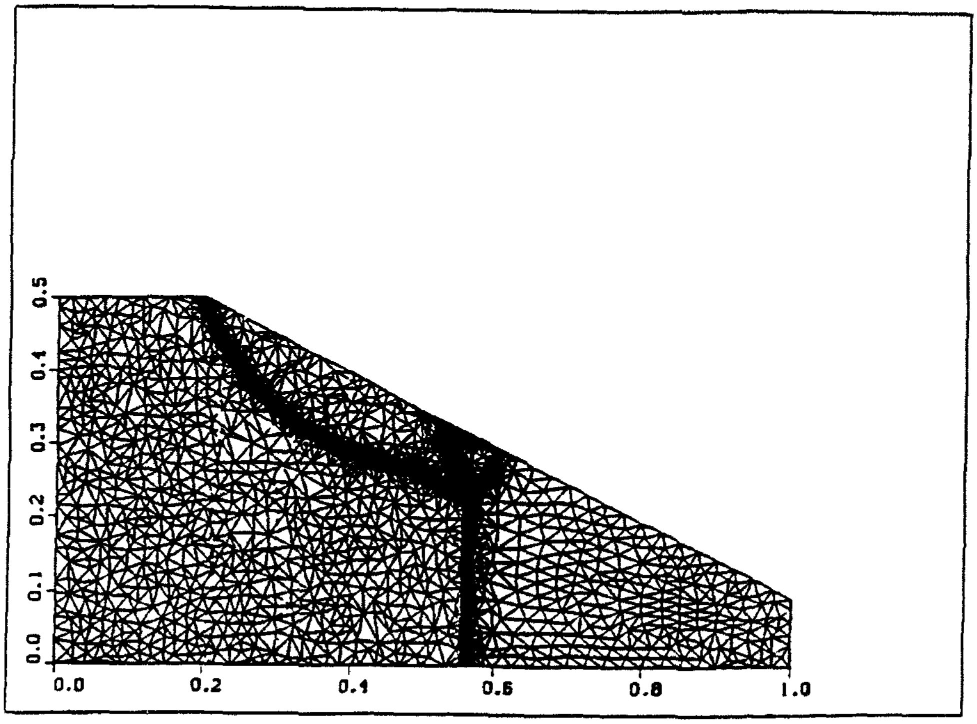


Figure 3.3.2b Grid at intermediate time for shock in planar channel
(M = 8.7, wedge angle = 27°).

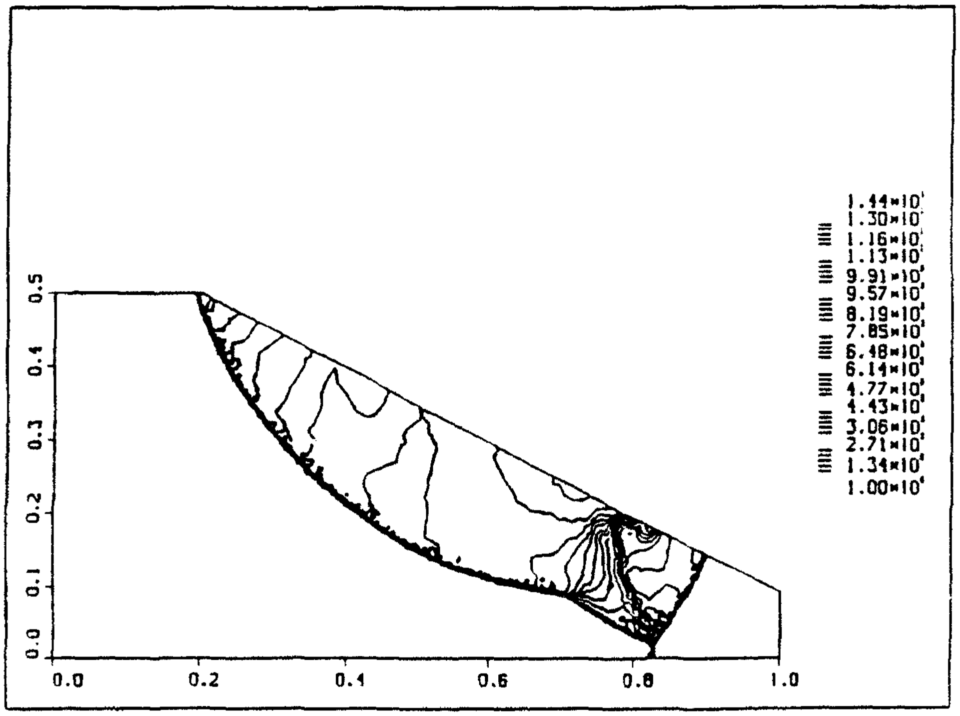


Figure 3.3.3a Density contours at late time for shock in planar channel
(M = 8.7, wedge angle = 27°).

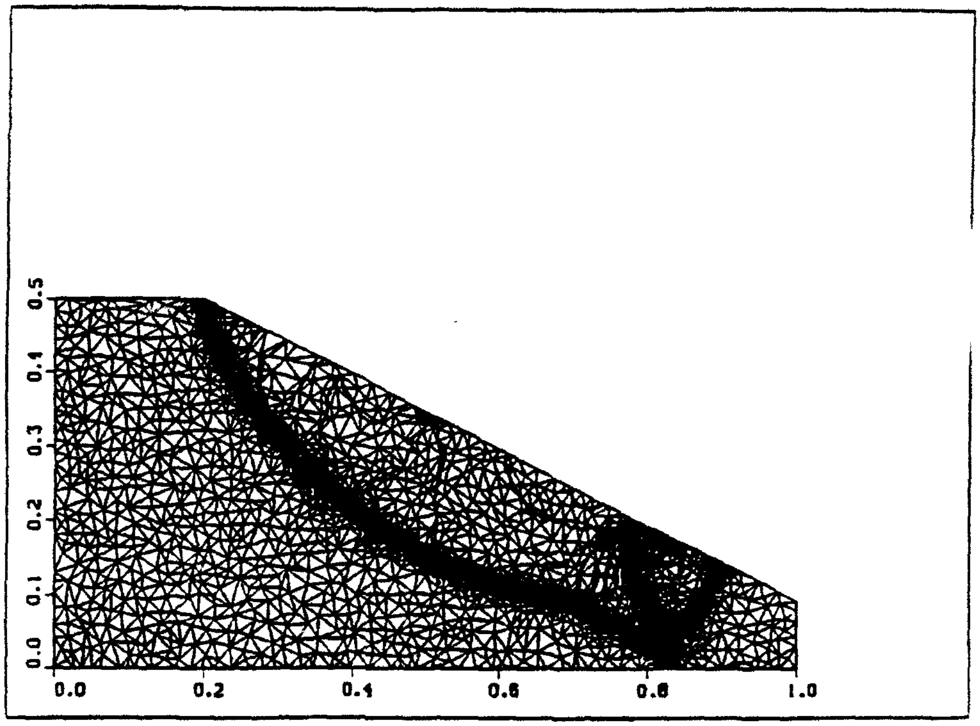


Figure 3.3.3b Grid at late time for shock in planar channel
(M = 8.7, wedge angle = 27°).

3.4 MINE EXPLOSION UNDER VEHICLE

The main objective of this joint Marine-Army program was the development of vehicles hardened against antitank (AT) land mines. The basic vehicle is the M925 5-ton cargo truck. Numerical simulations were used to determine the dynamic loads produced by the AT mine detonation on the cargo bed and other structural elements of the truck.

The algorithms, techniques and codes developed under the UUGM program provided two key elements necessary for the numerical simulations for this project: a) flexibility in describing the very complex geometry of the truck; b) high resolution-calculation of the shocks and other discontinuities using an adaptive unstructured grid. A version of the AUGUST-2D code developed under the UUGM program is being used for the analysis of blast resistance of different truck geometries.

We have carried out four such calculations, using four, eight, eight, and 20 pounds of C-4 explosive. These employed fixed (nonadaptive) meshes with 30,000 (4-1b case), 21,000 (8-lb cases).

A one- or two-dimensional calculation was performed to produce the initial blast profiles laid down on the three-dimensional grid. Aside from the amount of explosive, the calculations differed in the following ways: all but the 4-lb blast were centered beneath

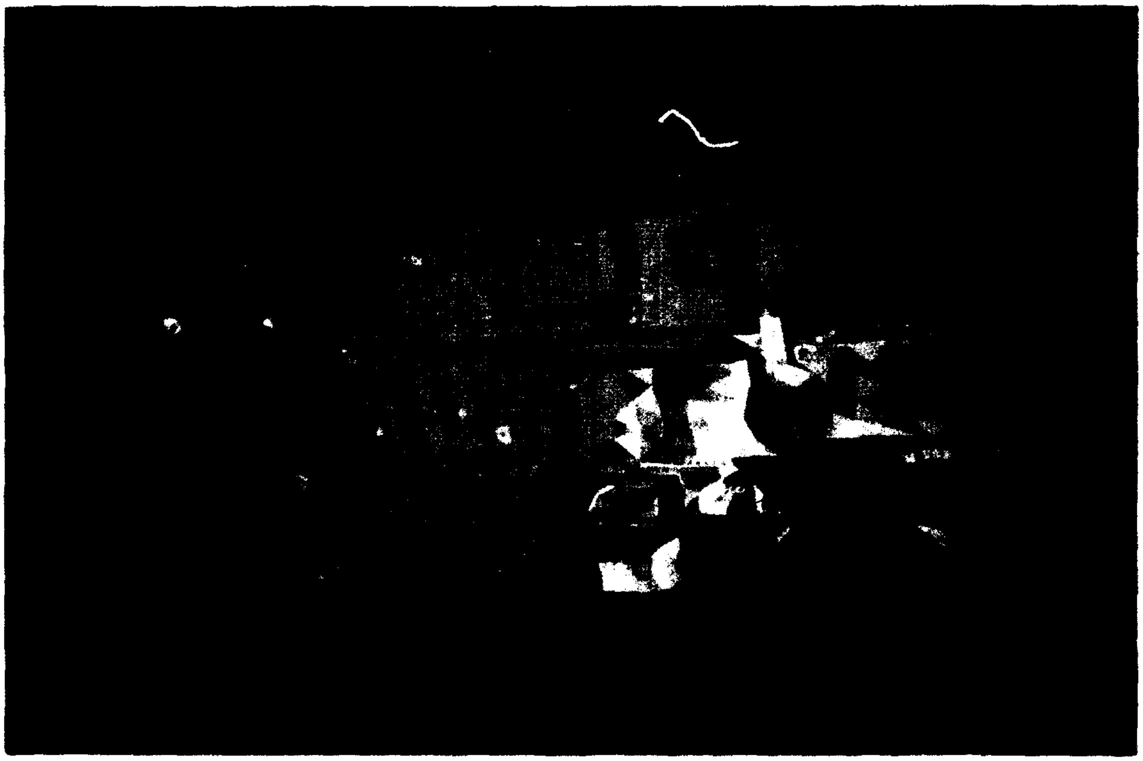


Figure 3.4.1 Two views of interaction between mine blast and M925 cargo truck: pressure contours at t = 0.574 msec.

the left front wheel of the truck (the 4-lb blast was situated 70 cm further back); for the first 8-lb case a crater with diameter 60 cm and depth 30 cm was situated underneath the blast.

All but the second 8-lb case used an ideal-gas equation of state with $\lambda = 7/5$ for :e air and detonation products. Twenty pressure "sensors" positioned on the mesh at pc .ts corresponding to the pressure gauges used in actual field tests were used to recorc the pressure and impulse histories there for comparison with the experimental data.

The calculations were run out to about 4.5 msec. The pressure stations closest to ground level and to the blast center exhibited peaks up to ~ 10^3 psi. In some cases multiple peaks were present, corresponding to reflected shocks.

An example of the domain decomposition of the computational grid for a typical mine-truck interaction problem is shown in Fig. 2.5.2. In Fig. 2.5.3 the unstructured triangular grid is used to describe a cross section of an M925 cargo truck. Use of unstructured grids allows detailed description of the truck geometry. Figure 3.4.2 shows results of the simulation in the form of pressure contours overlaid on the unstructured grid, viewed from two different directions halve a millisecond after the detonation.

At Ft. Belvoir's request, SAIC also assessed the damage to a mine-clearing plow due to a single detonation of an AT mine at close range during the Desert Storm operation. At that time, Ft. Belvoir RDEC had responsibility for support of countermine activity in the Desert Storm operation.

To simulate the plow-mine blast interaction, SAIC used computational capabilities partially developed under the UUGM program. Use of unstructured triangular grids again enables detailed description of the plow geometry and use of Direct Dynamic grid Adaptation method allows detailed simulation of the complex pattern of the shock wave reflections.

In Fig. 3.4.2 the initial stage of the blast-plow cross section interaction is shown in the form of pressure contours overlaid on the dynamically adapting grids. In Fig. 3.4.3 a more advanced stage of the blast-structure interaction is shown in the same format as in Fig. 3.4.2. The adaptive grid allows high resolution of a complex blast interaction phenomena.

SAIC has also simulated the structural response of the plow to the dynamic load that is defined by the gas dynamic simulations described above. In Figs. 3.4.4a - d results are shown for the plow deformation in response to dynamic load. Recent experimental assessment of the plow damage showed that SAIC predictions correctly described blast damage to the plow.

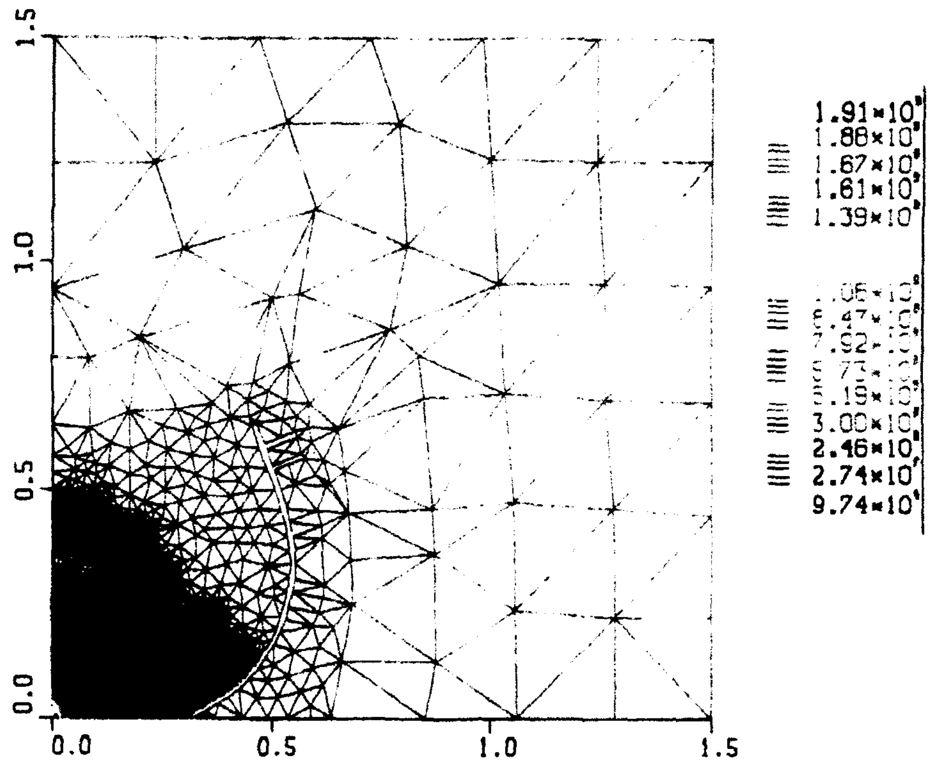


Figure 3.4.2 Blast – plow interaction: pressure contours in initial stage.

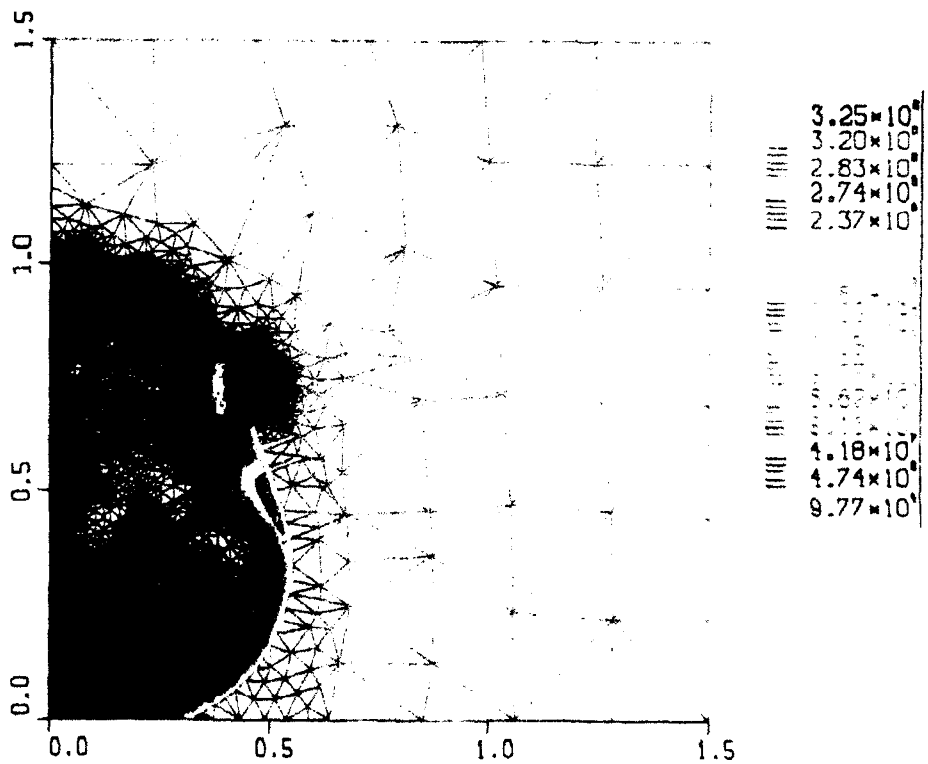


Figure 3.4.3 Blast – plow interaction: pressure contours in advanced stage

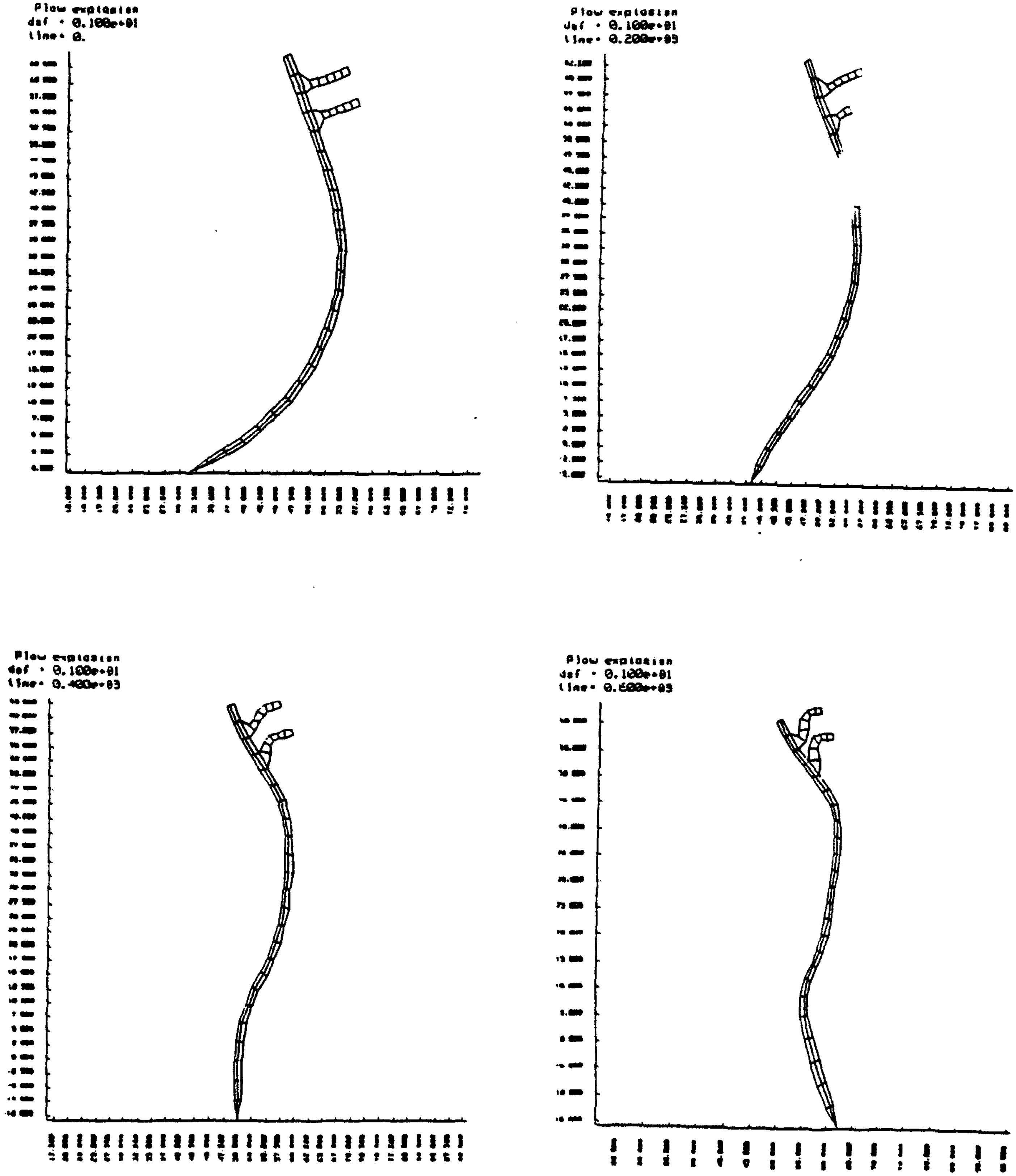
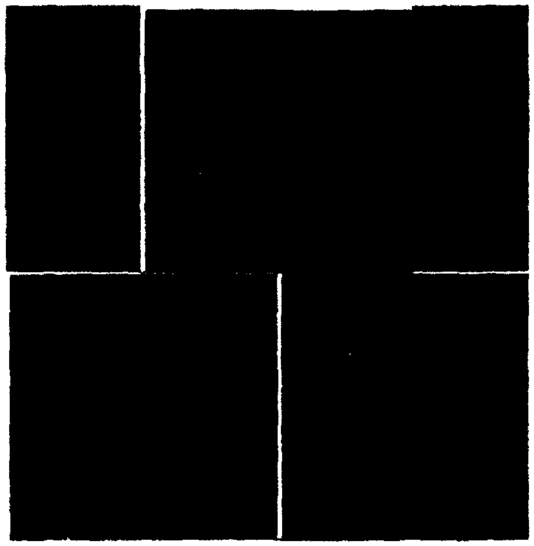
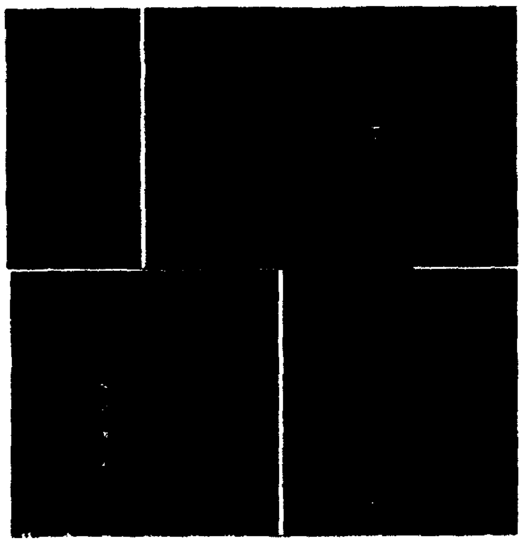


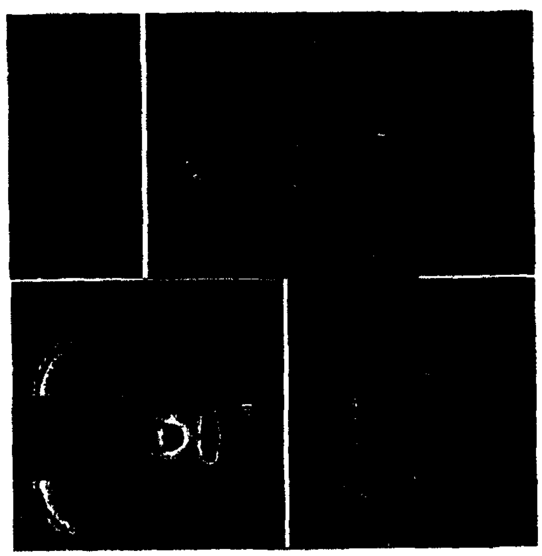
Figure 3.4.4 Structural response of the plow to blast load:
a) $t = 0$; b) $t = 200$ msec; c) $t = 400$ msec; d) $t = 600$ msec.



a. Predetonation stage



b. Detonation



c. Detonation product expansion

Figure 3.5.1 Pulsed detonation engine simulation: flow tracers.

3.5 PULSED DETONATION ENGINE

The main objective was the development of a revolutionary propulsion concept based on intermittent detonative combustion. Development of this concept will result in a new class of engines with performance surpassing those of small turbines at signific ::ly reduced cost. SAIC's PDE research was noted in a recent article [Aviation ' -ek, October 28, 1991, pages 68-89]. The PDE is currently considered as a candidate c :ncept for numerous propulsion systems including the air-to-air missile, cruise missile. RPV engine, high altitude UAVs and others.

The codes developed under the UUGM program have enabled SAIC to conduct a detailed study of the PDE concept. The unstructured grids used in the simulations allowed us to describe the complex geometries of the detonation chamber and air inlets for a full missile configuration. Adaptive gridding allowed efficient and accurate simulation of the detonation and resulting shock waves interacting with the thrust-producing surfaces of the engine.

In Fig. 3.5.1 results are shown for the simulation of the PDE detonation cycle for a Mach 2 missile. Lagrangian flow tracers are used to track air and fuel trajectories in the engine. The figures demonstrate the sequence of stages in one PDE cycle. Shown in Figs. 3.5.1a-c are the fuel mixing stage, the detonation stage and the detonation products discharge stage, respectively. Detailed CFD analysis of various geometries and flow regimes allowed us to develop an understanding of the parametric dependence of the fundamental variables that determine the PDE performance.

3.6 SHOCK FOCUSING IN AIR

Research relating to focusing of shock and acoustic waves is of considerable practical interest for application to extracorporeal shock-wave lithotripsy (ESWL). A schematic of the cross section of such a reflector is shown in Fig. 3.6.1. Strong acoustic waves are generated in the left focal point of the ellipsoid by an instantaneous release of energy and are refocused at the right focal point. Ideally, focusing should be based on waves of acoustic intensity, since the nonlinear reflections of strong shock waves lead to significant distortions in wave propagation and impair simple geometrical focusing.

Figure 3.6.1 shows the computational domain and grid for the ellipsoidal reflector. Figure 3.6.2 shows the simulation results at time $t = 1.21 \times 10^{-6}$sec. At this stage, the wave front that propagated to the left has undergone full reflection and the reflected wave propagates in the direction of the incident wave to the right. Figure 3.6.3 shows the pressure contours ($t = 8.41 \times 10^{-4}$sec) when the maximum focused pressure is obtained in the system. The incident front has left the computational domain, and the maximum pressure occurs in a small volume in the vicinity of the right focal point. The maximum focused pressure has reached 1.37×10^5 Pa and is located 11 mm to the right of the focal point of the ellipsoid. In all the figures presented, the method of composite domain decomposition works extremely well, producing seamless solutions at the interfaces.

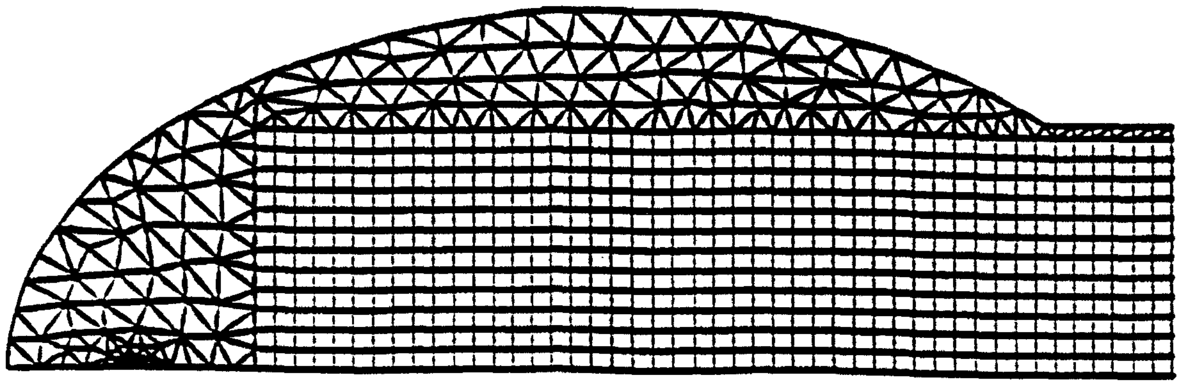


Figure 3.6.1a Hybrid structured/unstructured grid used for numerical simulation of ellipsoidal reflector.

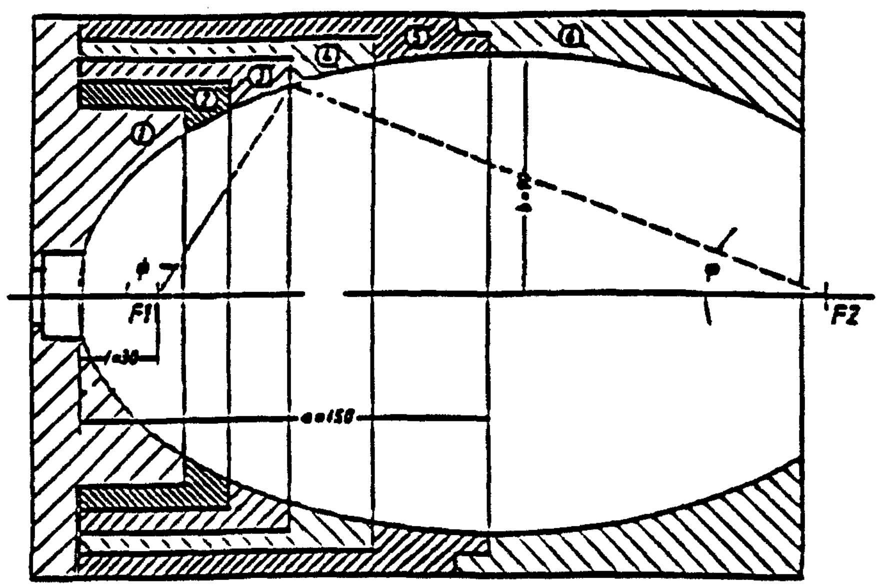


Figure 3.6.1b A schematic drawing of the center cross section of the ellipsoidal reflector.

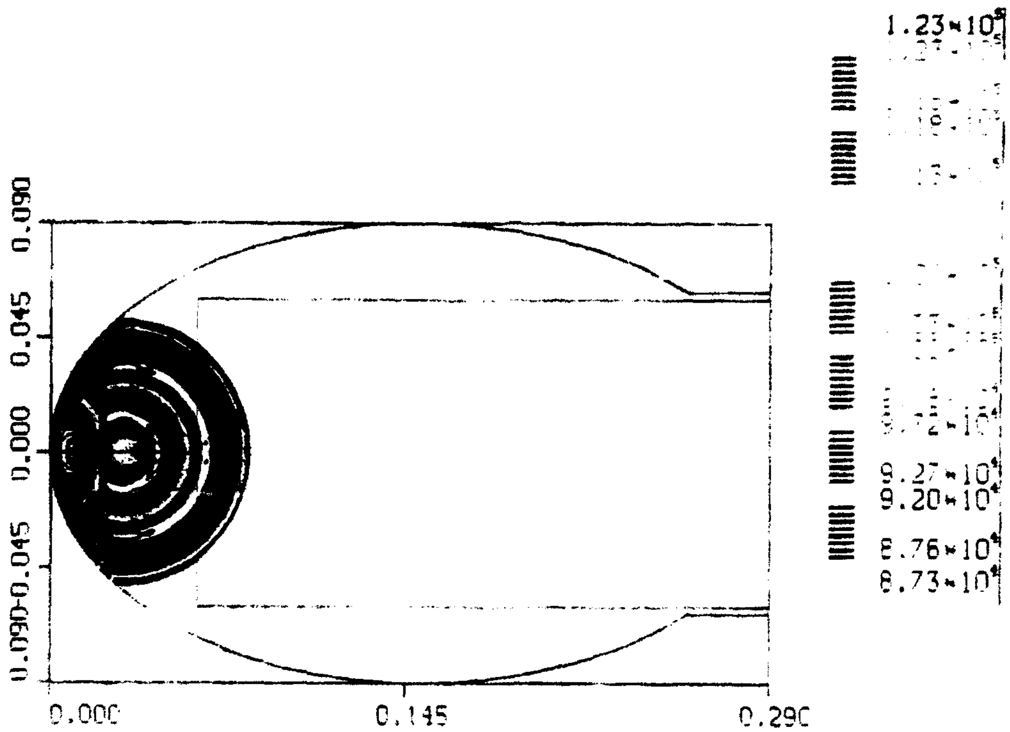


Figure 3.6.2 Pressure contours at time t = 1.21 × 10^{-6}sec showing the incident wave as reflected from the reflector wall.

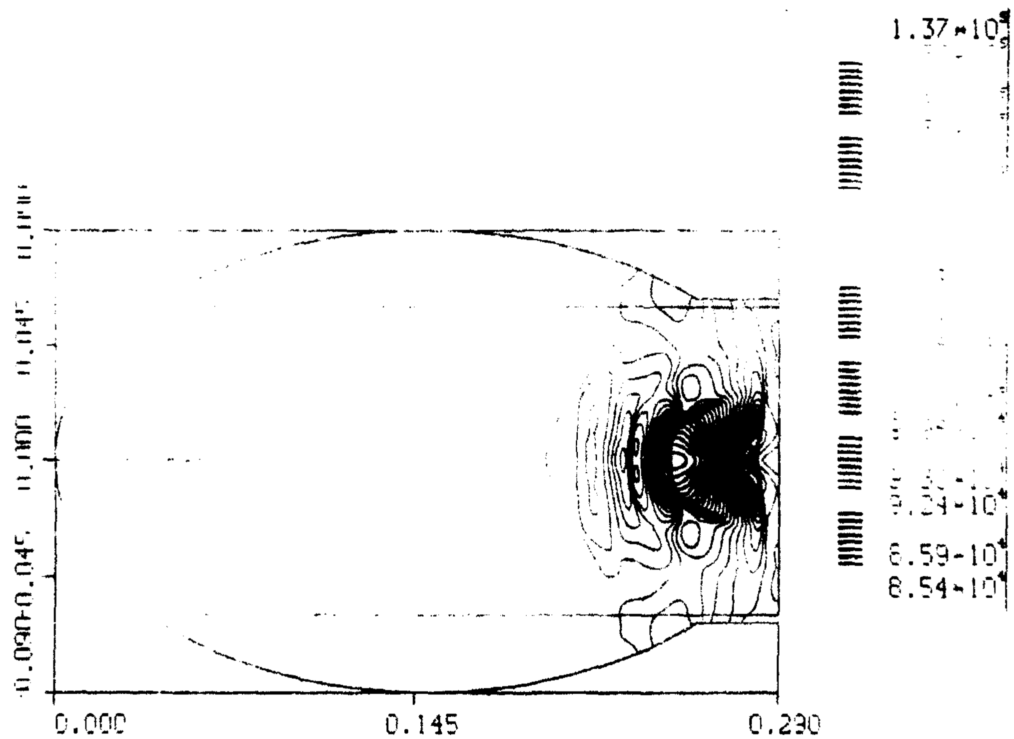


Figure 3.6.3 Pressure contours at time t = 8.41 × 10^{-4}sec showing the stage at which the maximum focused pressure is obtained

3.7 NONIDEAL AIRBURST IN MULTIPHASE MEDIA

The main objective was to advance the understanding of the formation dynamics and microphysics of the multiphase flow of clouds developing as a result of a nuclear explosion. A main difficulty in analysis of nuclear cloud formation is the necessity to take into account physical phenomena that are interdependent and occur on vastly different scales. At about 30 seconds after a nuclear detonation, the cloud can be 4 km high and the shock wave will be at the distance of 10 km. The multiphase interactions that occur on a scale of 10-100 meters are very important and have to be accounted for.

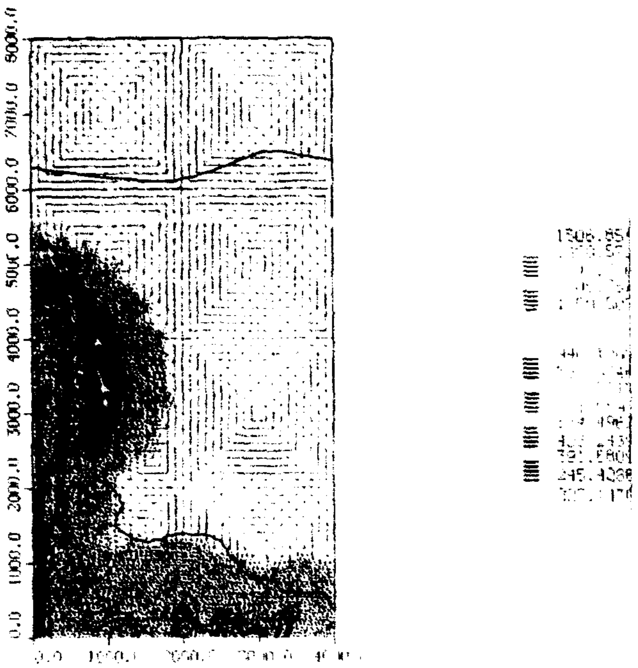
SAIC has developed a multiphase, multicomponent version of the AUGUST-2D code developed under the UUGM program. We use an explicit method for the solution of the multiphase flow described by equivalent Euler equations, and an implicit integration for simulation of the particle-fluid interactions. The grid adaptivity allows efficient and accurate simulation of this multiphase phenomenon. The grid adaptivity is used for adjusting the spatial scale of the domain decomposition to the scale required for accurate simulation of various physical interactions. Other code improvements such as introduction of the real-gas equation of state and Lagrangian particle tracing were employed to enable simulations and analysis of this complicated phenomena.

In Fig. 3.7.1a the computational domain and grid are shown for the nuclear cloud simulation. In this figure the temperature contours are overlaid on the unstructured grid. In Fig. 3.7.1b the particle density contours are shown for the same stage of the cloud evolution as in Fig. 3.7.1a. In Fig. 3.7.1c particle radius is shown for the same stage of the cloud evolution, and Fig. 3.7.1d shows locations of the Lagrangian tracers that mark evolution of the detonation products.

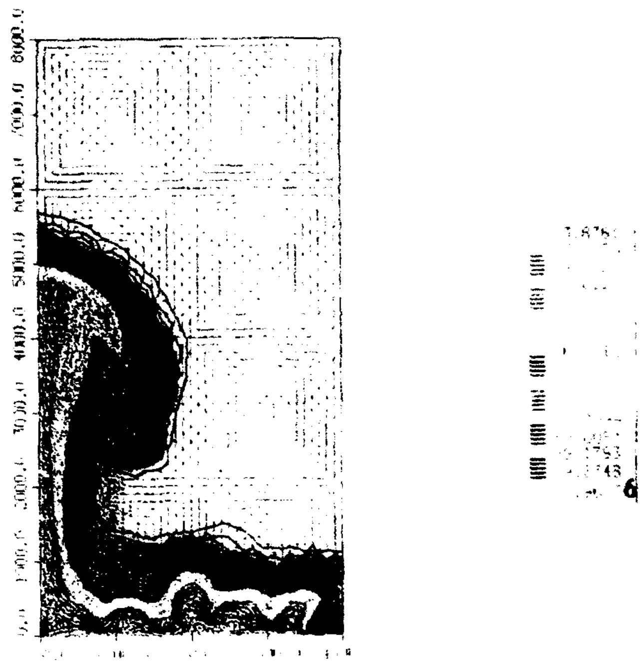
3.8 FLOW IN THE SARL WIND TUNNEL

One of the problems to which AUGUST 3D has been applied is that of modeling the SARL wind tunnel at Wright Laboratory. This example is a good test of the use of the Second-Order Godunov method to do nearly incompressible flow calculations. To illustrate the results, Fig. 3.8.1 shows the grid used for simulating the flow. The calculation was performed by specifying the inflow and outflow parameters and running the simulation to convergence. The run was performed at SAIC on the Stardent workstation and repeated on an Iris at FIMM. Figures 3.8.2 and 3.8.3 show the pressure levels in the tunnel. The results were visualized using AVS.

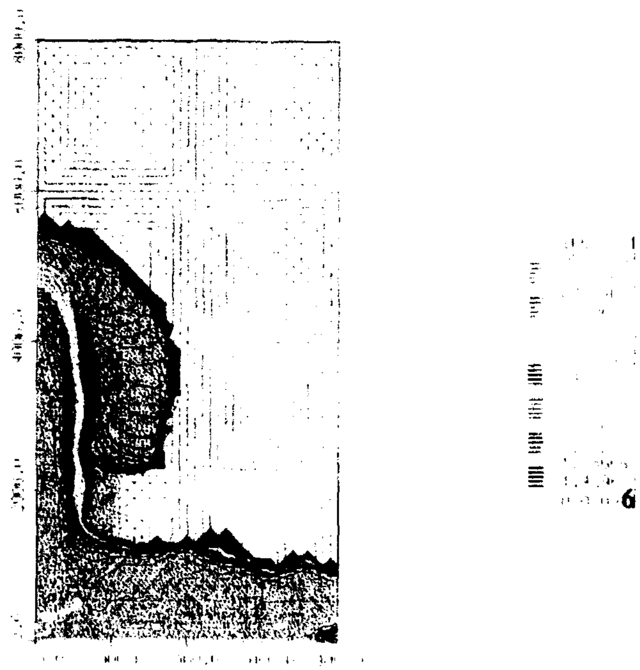
Figures 3.8.1 and 3.8.2 show two views of the pressure contours generated in a calculation of subsonic flow (Mach number 0.05). The results were confirmed by comparison with those obtained using a code with a structured grid, and by checking them against measurements.



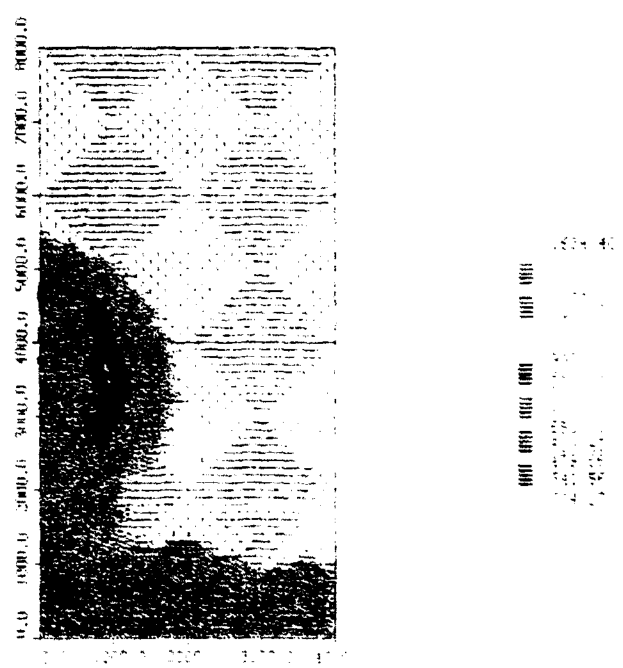
a. Temperature contours and grid.



b. Particle density.



c. Particle radius.



d. Lagrangian traces.

Figure 3.7.1 Formation of a radiative cloud. Multiphase simulation.

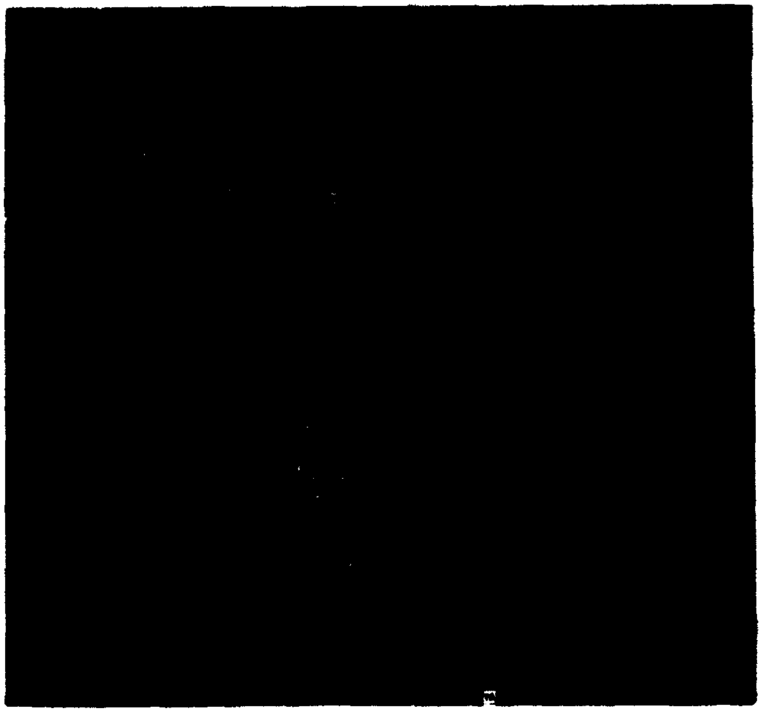


Figure 3.8.1 The unstructured grid used to simulate the SARL wind tunnel.

Figure 3.8.2 The pressure contours from the simulation of the SARL wind tunnel.

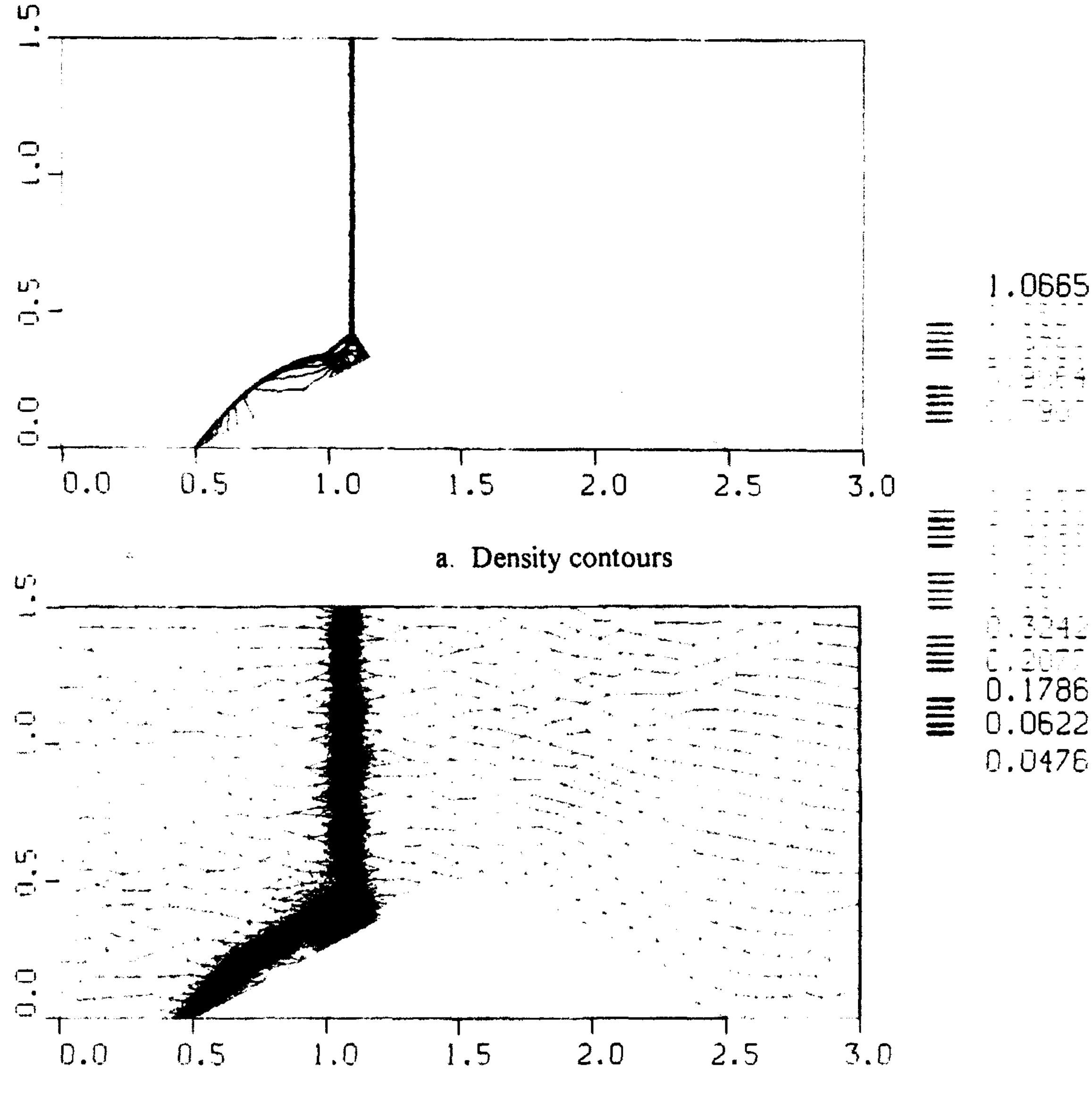
3.9 SHOCK ON DOUBLE WEDGE

A much more complicated problem, which has been extensively studied to benchmark and validate Euler solvers, is flow over a double wedge. This problem contains multiple fluid phenomena and is a stringent test for any solver. It includes shock formation, a Mach stem, rarefaction, a slip line, vortex generation and rollup, and is transient in nature. To validate our direct dynamic refinement method in AUGUST, we simulated a Mach 2.85 shock wave propagating in a channel and impinging on a symmetric 45° wedge, and also a Mach 8.7 shock impinging on a symmetric 27° wedge.

Both of these compared well with experimental results. Figure 3.9.1 shows an interferogram taken from Glaz et al.,[17] showing the M = 8.7 shock interacting with the front surface of the 27° wedge. Our results are shown in Figs. 3.9.2 - 3.9.4. The first of these illustrates the grid and density when the shock is on top of the wedge. The shock is well resolved and the grid is well adapted in the vicinity of important features and coarsened in the region that the shock has passed through. The next two figures show the evolution of the flow and the grid after the wedge where comparison can be made with the experimental results. AUGUST produces no artificial features and recovers the phenomenology seen in the experiment.

Figure 3.9.1 Experimental interferogram of a shock hitting a 45° corner at $M_S = 2.85$.

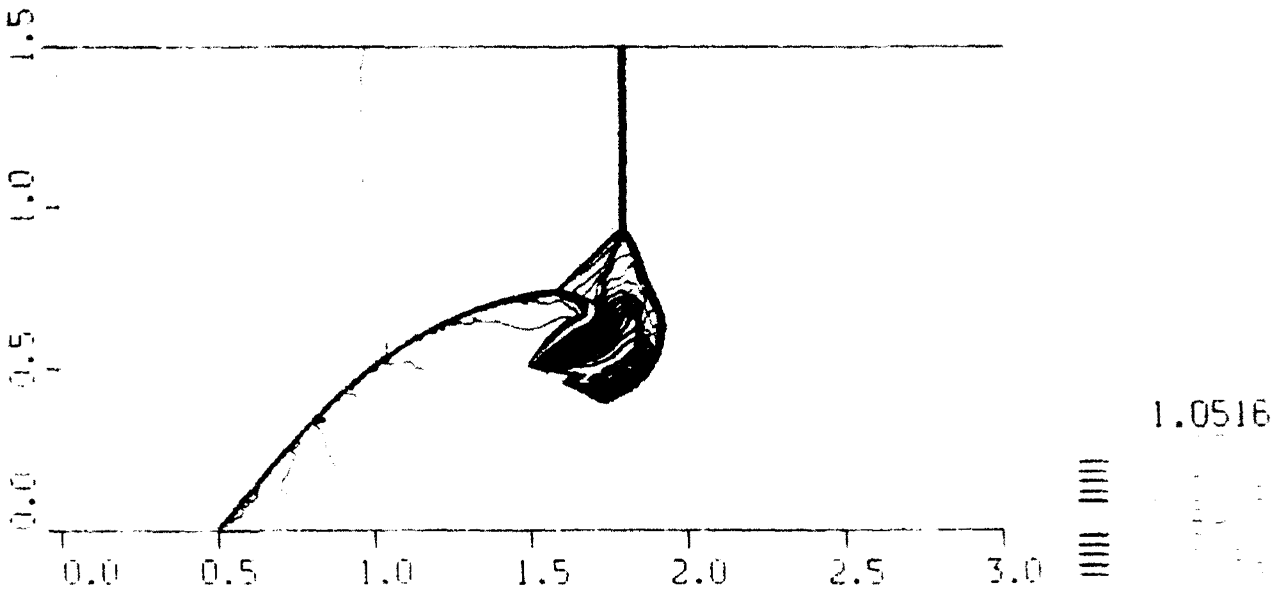
In the figures showing the triangular grids, the area of a triangle in the dense region of the grid is roughly 100 times smaller than the area of a triangle in the initial grid. The figures show that the grid adaptivity is capable of capturing the flow gradients including shocks, contact discontinuities and slip lines. Formation of a triple point of the Mach reflection, slip line and strong vortex formation are seen in Fig. 3.9.2a. In fairness, most of the flow phenomena that is captured by AUGUST have also been captured by other CFD schemes.[18] However, the accuracy estimated for the AUGUST numerical calculations in this example is on the order of 4%, equal to the accuracy of the experimental observations.



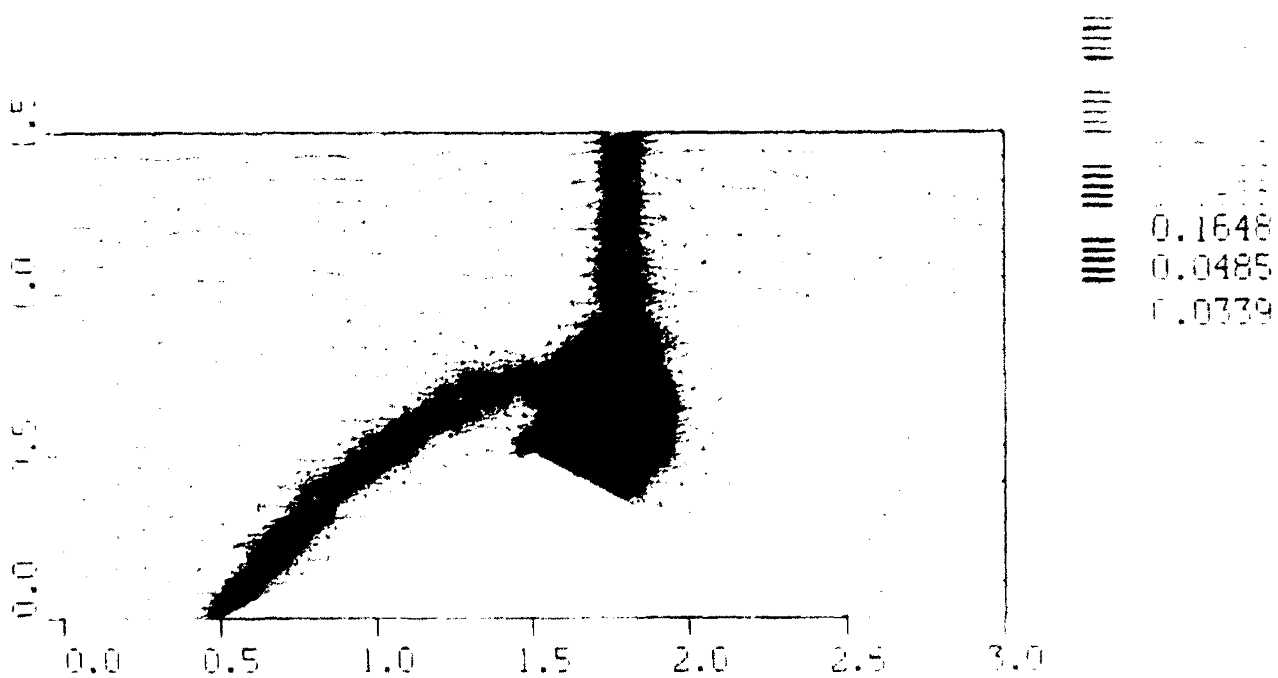
a. Density contours

b. Adaptive grid, 15,123 vertices

Figure 3.9.2 Interaction of a Mach 8.7 planar shock wave with a 27° double ramp: Mach reflection stage.

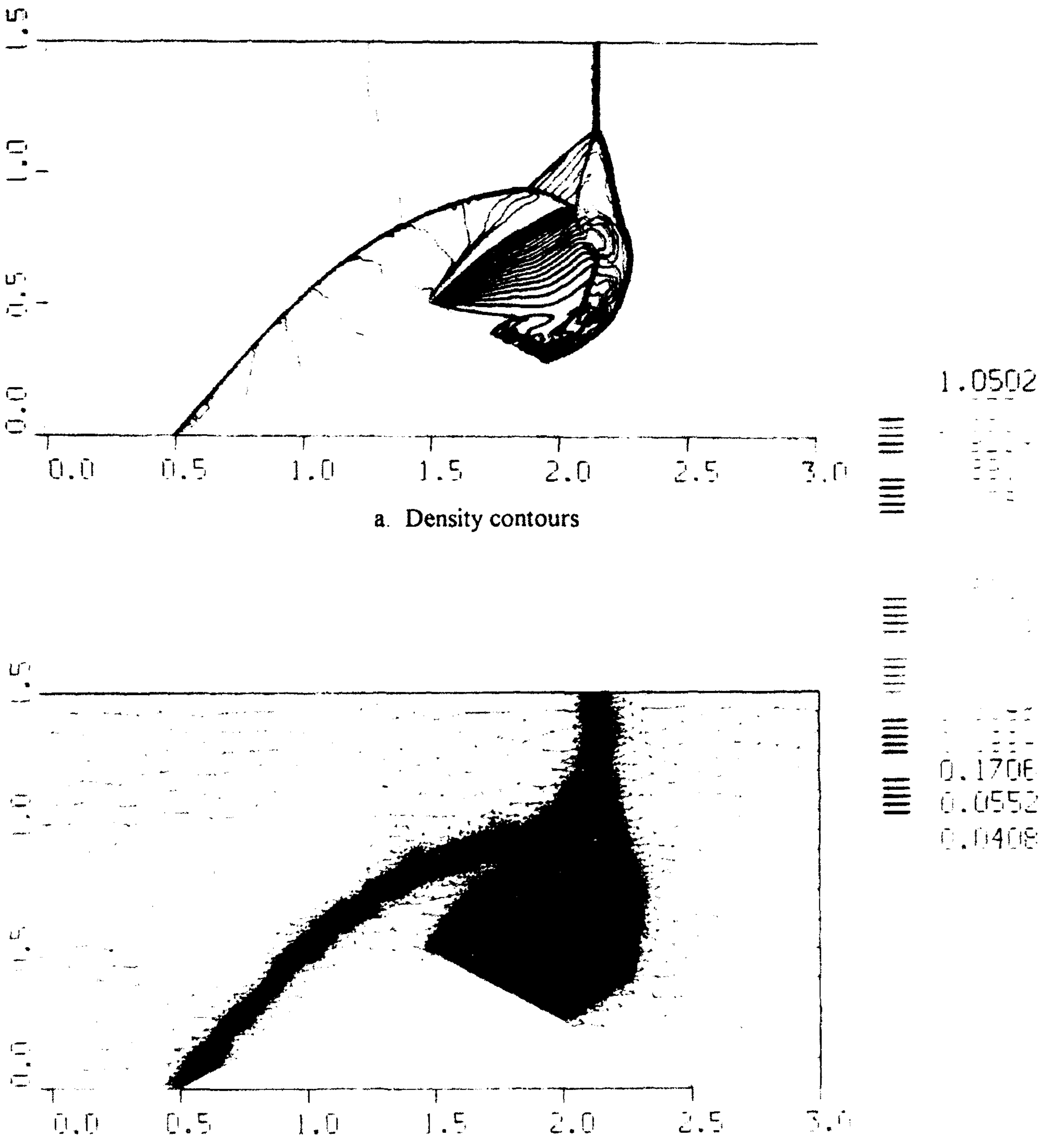


a. Density contours



b. Adaptive grid, 27,132 vertices

Figure 3.9.3 Interaction of Mach 8.7 planar shock wave with a 27° double ramp: start of the diffraction stage.



a. Density contours

b. Adaptive grid, 46,462 vertices

Figure 3.9.4 Interaction of Mach 8.7 planar shock wave with a 27° double ramp: shock diffraction stage.

3.10 SUPERSONIC SPRAY COATING DEVICES

In this section we present the results of an application of the UUGM sim‿ ›n technology to a sample problem involving spray-coating devices. Here we only trc .ıe aerodynamic flow of particles in a high temperature gas which is moving supersc ly: we consider a reasonably complex geometry including a simulated surface tha: the substrate to be coated. The details of the surface interaction resulting in depos:: .ı are not treated in this example.

In Fig. 3.10.1 the computational domain and grid are shown for a model supersonic jet sprayer device that includes reactor nozzle, solid particle injector, and expansion nozzle. Also shown in Figure 3.10.1 is a perforated flat surface substrate placed in the flow field. The high-velocity high-temperature flow stream exiting the reactor nozzle accelerates the injected particles. The particles are heated during acceleration, melt, then expand with the flow in the nozzle, gain more speed, and finally impinge onto the surface. Details of the flow-surface interaction (here without boundary layers taken into account) will strongly affect the uniformity with which the surface will be "coated" by the particles carried by the flow.

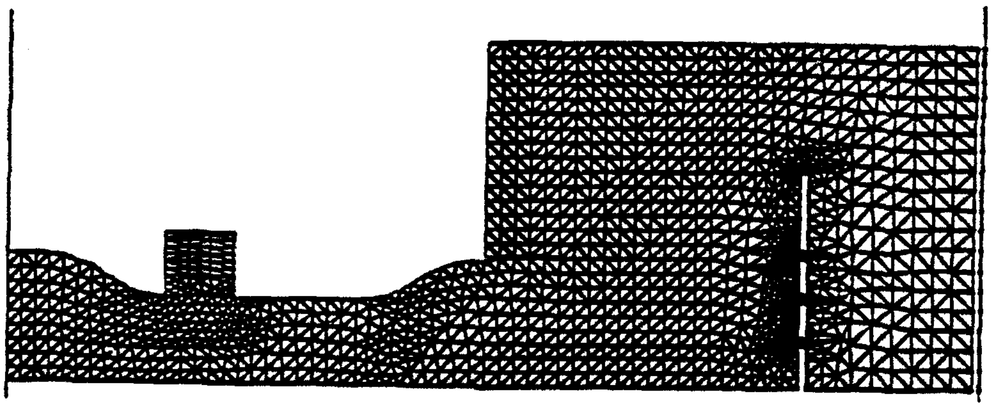


Figure 3.10.1 The figure shows the initial computational grid for the jet spray simulation demonstr`ıtion. Shown are the nozzle, injection region and target surface depicted as a flat plate with perforations, oriented perpendicular to the mean spray flow. The boundary conditions used for the sample simulation were: V_g = 1000 m/sec, ρ_g = 0.1 kg/m^3, T_g = 3500 K at the inlet of the reactor nozzle; V_g = 1500 m/sec, ρ_g = 0.3 kg/m^3, T_g = 1500 K, V_p = 1500 m/sec, T_p = 1500 K, N_p = 2000 at the inlet of the reactor nozzle.

To trace the motion of the particles in the plasma spray device and the interaction pattern with the target surface we have injected Lagrangian "marker" particles (massless but moving with the local flow speed) in the particle injector flow stream. In Fig. 3.10.2 results are shown in the form of marker particle locations. To monitor the particle temperatures we have introduced particle coloring, where the color defines the local particle temperature. Thus one can evaluate the evolution of the particle temperature by observing the particle color transition. This coloring scheme can be used to show other parameters such as particle residence time or density. This represents a simple method of visualization that we have used successfully in past UUGM simulations.

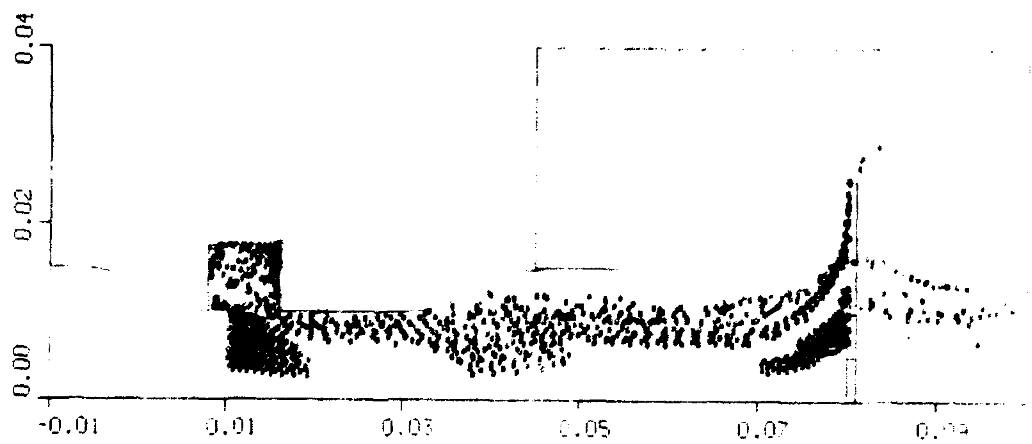


Figure 3.10.2 Lagrangian marker particles are shown in color representing the evolution of injected particle temperature as a function of particle position and time in the jet spray stream.

In Fig. 3.10.3 simulation results for the steady state are presented in the form of gas temperature contours for the jet spray system. Here it is possible to observe a very large temperature variation in the nozzle. The cold gas that is injected with the particles remains at the edge of the jet stream. At the same time the main jet cools through the expansion in the nozzle from 3500°K to 2000°K, and then undergoes a series of expansions and compressions in the system of shock waves created by overexpansion of the supersonic jet. Figure 3.10.3 also shows a nonuniform temperature distribution on the surface that is partially created by the gas flow through the perforated holes.

In Figs. 3.10.4 and 3.10.5 simulation results are shown for the density and pressure contours. Here we can observe the formation of several diamond-shaped shock structures as a result of supersonic flow over expansion. However, for the flow regimes in our simulation these shocks do not lead to a higher rate of mixing by injected cold gas with particles and the main hot gas stream. This can be noticed in the density contours, where one clearly observes that the high-density cold gas does not penetrate the main hot jet flow. By changing the condition (injection pressure, angle of entry, etc.) of the injected flow one can improve mixing, thus achieving higher particle temperatures and velocity.

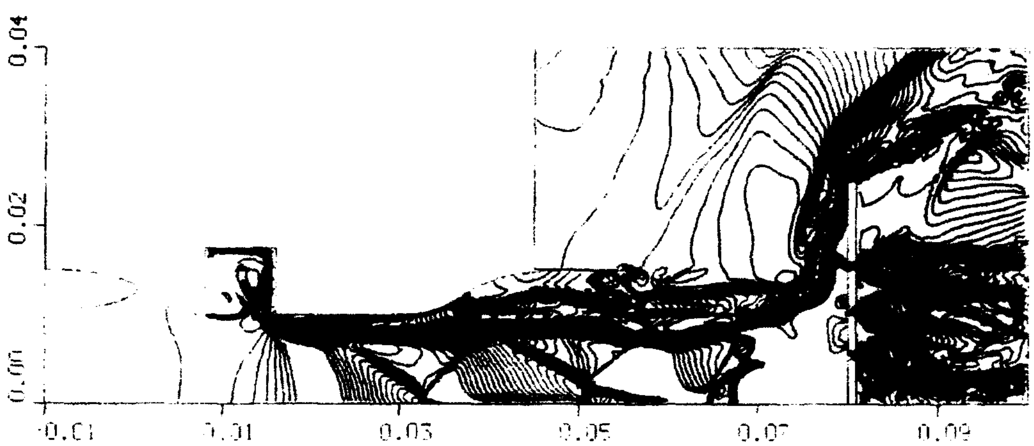


Figure 3.10.3 Gas temperature contours in the jet spray stream. The maximum temperature is 3500°K and the minimum is 600°K.

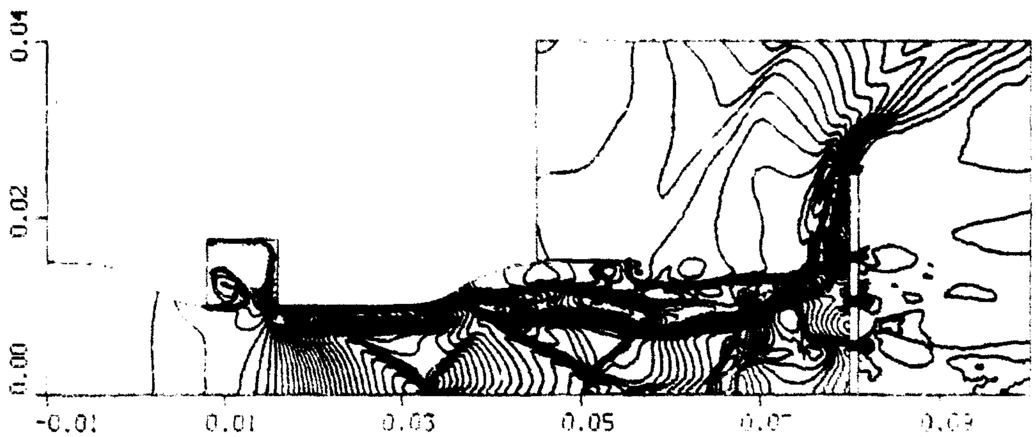


Figure 3.10.4 Gas density contours in the jet spray stream. The injected stream and the main flow mix poorly. The diamona patterns describe the shock wave pattern resulting from the flow's overexpansion.

Figure 3.10.5 Pressure contours in the jet spray stream. The diamond patterns show that supersonic flow is maintained near the vicinity of the target surface

3.11 DUSTY FLOW OVER A CYLINDER

A numerical study of two-phase compressible flow has been performed for the reflection and diffraction of a shock wave propagating over a semicircular cylinder in a dusty gas. The following model was used to derive the governing equations:

(1) The gas is air and is assumed to be ideal;

(2) The particles do not undergo a phase change because for the particles considered here (sand) the phase transition temperature is much higher than the temperatures typical for the simulated cases;

(3) The particles are solid spheres of uniform diameter and have a constant material density;

(4) The volume occupied by the particles is negligible;

(5) The interaction between particles can be ignored;

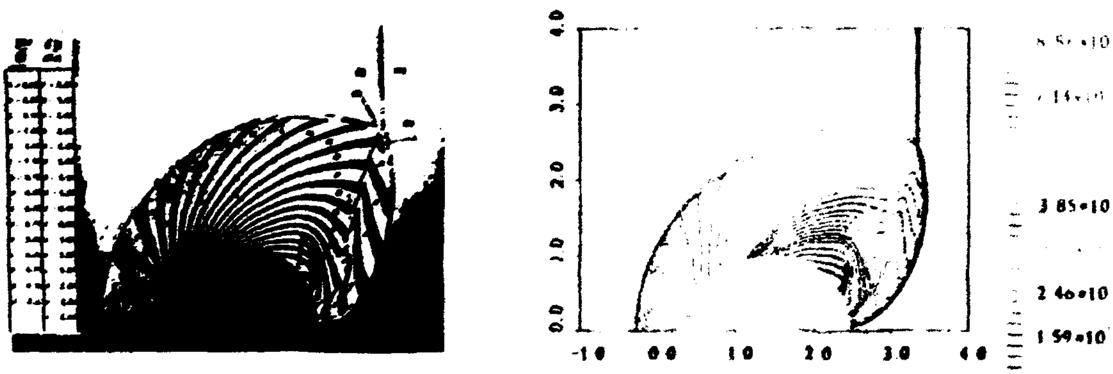
(6) The only force acting on the particles is drag and the only mechanism for heat transfer between the two phases is convection. The weight of the solid particles and their buoyant force are negligibly small compared to the drag force;

(7) The particles have a constant specific heat and are assumed to have a uniform temperature distribution inside each particle.

Under the above assumptions, separate equations of continuity, momentum, and energy are written for each phase. The interaction effects between the two phases appear as source terms on the right-hand sides of the governing equations. The two phases are coupled by interactive drag force and heat transfer.

The objectives of the study were (a) to solve the two-phase compressible flow field and compare the simulation with available experimental results; (b) to observe and investigate the reflection and diffraction wave patterns when a shock wave propagates over a semicircular cylinder in a dusty gas, with particle radius and loading as parameters.

To test the accuracy of the two-dimensional computation, we first computed the pure gas flow case of shock wave reflection and diffraction over a semicircular cylinder. We then compared the simulation with experimental results. Shock wave reflection on a wedge has been extensively studied by many researchers (see e.g., review papers of Ben-Dor and Dewey[18] and Hornung.[19] As one can see from Fig. 3.11.1, the results show excellent quantitative and qualitative agreement between the numerical simulation and experimental results.



(a) (b)

Figure 3.11.1 Comparison for $M_s = 2.8$ pure-gas flow: (a) interferogram from experiment; (b) density contours from present calculation.

In the two-phase simulation a planar shock with $M_s = 2.8$ propagates into an area of dusty gas and impinges on a semicircular cylinder. The interface between pure air and dusty air is located at $x = 0.0$ of the computational domain. The area of the dusty air contains a semicylinder with a radius of 1m. The size of the computational domain, initial parameters of the gas, parameters of the incoming shock, size of the semicylinder and its location in the computational domain, are the same as in the reflection and diffraction simulation in the pure gas case. The main objective of this set of simulations was to study the effects of particle size and particle loading on the parameters of the reflected and diffracted shock waves.

The first set of simulation results is shown for the case with dust parameters r_p = 10μm and ρ_p = 0.25 kg/m^3. The gas parameters and the parameters of the incoming shock wave were the same as in the pure gas case presented above. In Figs. 3.11.2a and 3.11.2b, the particle density and gas density contours are shown at the stage where significant diffraction has taken place and the shock front is approaching the trailing edge of the cylinder. To study the influence of particle loading on the dynamics of reflection and diffraction, we have simulated the case with a dust density of ρ_p = 0.76 kg/m^3 and with r_p = 10μm. To examine the effect of particle size on the reflection-diffraction process, we simulated a case where the particle loading and gas flow conditions were the same as in the previous case with particle density ρ_p = 0.76 kg/m^3, but the particle size was r_p = 50μm (Fig. 3.11.3).

On the basis of these calculatrions we reached the following conclusions:

(1) For a two-dimensional pure-gas flow, numerical results agree well with existing experimental data qualitatively and quantitatively, indicating that the gas phase is accurately simulated by the adaptive grid technique;

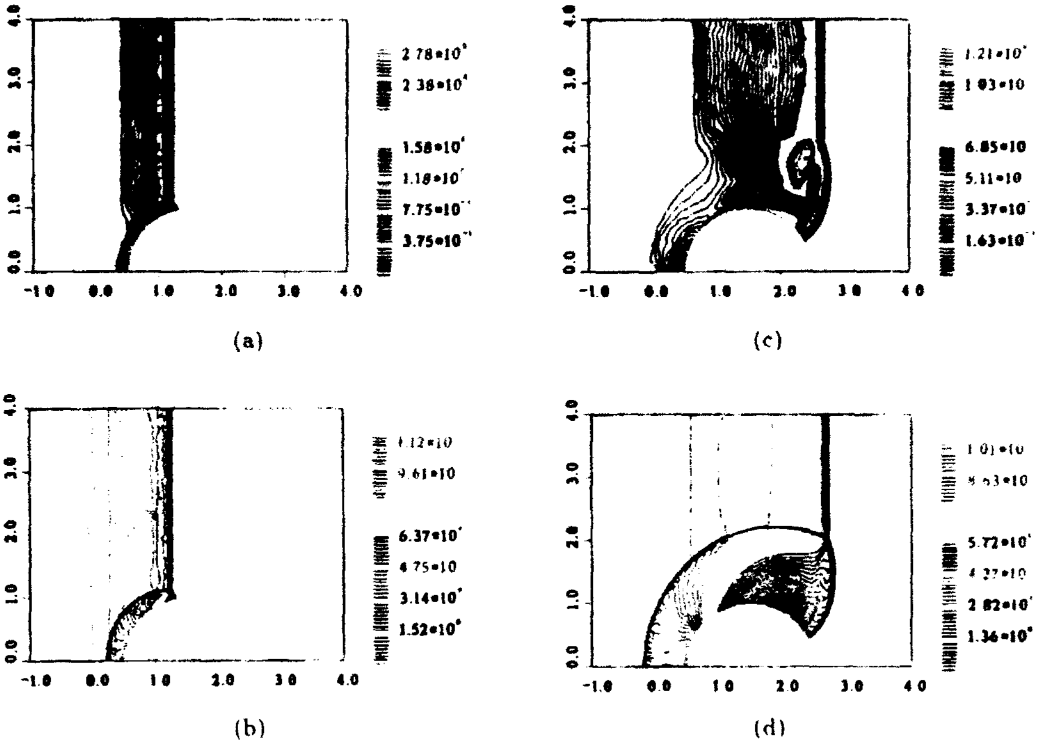


Figure 3.11.2 Density contours for the case $M_s = 2.8$, $\rho_p = 0.25$ kg/m^3, $r_p = 10\mu$m at two different times: (a) particle density at t_1, (b) gas density at t_1; c) particle density at t_2, (d) gas density at t_2.

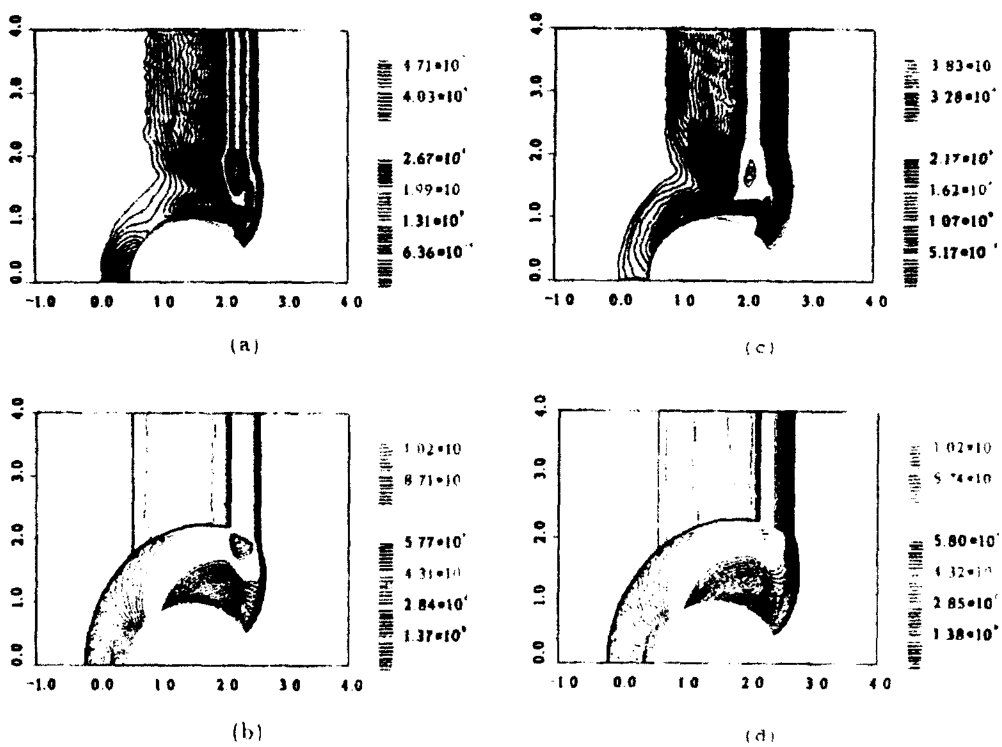


Figure 3.11.3 Density contours for the case $M_s = 2.8$, $\rho_p = 0.76$ kg/m^3, for two different particle sizes: (a) particle density and (b) gas density for $r_p = 10\mu$m; c) particle density and (d) gas density for $r_p = 50\mu$m.

(2) Particles in the gas can have a profound effect on the shock wave reflection and diffraction pattern, which is a function of particle size and loading. The ss the particle loading, the less the influence of particle on the flow field;

(3) In the three simulation cases, particles accumulate behind the "back ulder" of the semicircular cylinder due to the effect of particle inertia and the rarefaction ave;

(4) For different particle sizes at fixed particle loading, the larger particle will have a longer relaxation zone and less accumulation at the "back shoulder" and behind the incident shock. The gas density contours show a less distinguishable slip line in the small particle case than in the large particle case.

3.12 IMAGE PROCESSING

Very recently, there have been exploratory efforts in image processing based on nonlinear methods. If the purpose of an enhancement process is to highlight the edges of an image, then the technique used in the frequency domain is usually highpass filtering. An image can be blurred, however, by attenuating the high-frequency component of its Fourier transform. Since edges and other abrupt changes in the gray levels are associated with high-frequency components, image sharpening can be achieved in the frequency domain by a highpass filtering process, which attenuates the low-frequency without disturbing high-frequency information in the Fourier transform. The primary problem with this technique is that an ideal discontinuity has an infinite spectrum of frequencies associated with it. When filtering is applied, some frequencies are cut off, leading to a loss of edges in the image.

In computational fluid dynamics (CFD) similar problems exist in simulating flows with discontinuities. The problem of simulating flows with discontinuities is less forgiving, since an incorrect calculation usually leads to a complete distortion of the flow field. This has led CFD scientists to develop sophisticated algorithms that identify and preserve discontinuities while integrating the flow field in the computational domain. In the image domain, sharpening is usually done by differentiation. The most commonly used methods involve the use of either gradients or second derivatives of the pixel information. Central differencing is usually used to calculate the derivatives. CFD research has shown that this strategy will lead in many cases to smearing of the flow discontinuities (analog of the image edges in image enhancement).

A new and unique image sharpening method based on computational techniques developed for AUGUST has been developed. Preliminary experience shows that it can enhance image edges and deconvolve images with random noise. This indicates a potential application for image deconvolution from sparse and noisy data resulting from measurements of backscattered laser-speckle intensity.

The Second-Order Godunov Method used in AUGUST was developed from an understanding of the phenomenology of signal propagation in gasdynamical systems. The numerical algorithm implementing this method is not analytical and contains a set of steps that can be regarded as wave filters. These filters are designed to not smear the discontinuity (edge), suppress the spurious oscillations, and propagate the relevant signals through the system. The following algorithmic steps are performed to advance the solution for a single iteration in the Second-Order Godunov Method:

1. Local Extrapolation
2. Monotonicity Constraint
3. Characteristics Constraint
4. Riemann Problem Solution
5. Integration

Most of these steps have an analog in conventional image processing methods. Here we will give an explanation of the function of each algorithmic step of the Second-Order Godunov Method and where applicable, will point to its possible analog in conventional signal processing techniques.

Step 1 consists of extrapolation of the values in the computational grid (pixel) cell to the edges of the cell. Linear or nonlinear extrapolation can be used. This step is analogous to the standard edge-sharpening techniques used in image processing, with one important difference: the extrapolation is done not for the value itself but for its flux (change of value across cell boundary).

Step 2 includes a monotonicity constraint for the values at the cell edges. This is analogous to the nonlinear technique of locally monotonic regression only recently introduced for signal processing.

Step 3 subjects the values at the edges to the constraints derived from a solution of the one-dimensional characteristics. This step assures that the values at the edges have not been extrapolated from directions inconsistent with the characteristic solutions. This prevents extrapolation as well as smearing or overshoot of the discontinuities. For the image-processing application, this can be regarded as a form of automatic edge detection step where the shock waves are associated with the edges of an image.

Step 4 uses an exact solution of the system of the gasdynamic equations for calculation of the flux values based on the extrapolated values of the parameters at the left and right side of the edges. This step has no analogy in image processing. However, since the analytical solution includes discontinuities, an exact calculation of the flux at the edge location is allowed, even if this flux is calculated through a discontinuity.

Step 5 consists of finite-volume integration of the system of conservation laws. Here, the image is effectively treated as a flow field: the flux integration serves as a smoothing filter from the image perspective.

The effect of these steps is equivalent to the application of a unique filter stack with proven properties of discontinuity preservation and robustness.

The field of gray scale intensity of an image can be translated into a flow field. To every image pixel we assign to the corresponding cell of the computational domain values of the gasdynamical parameters proportional to the values of the gray scale. Our understanding of the basic gasdynamical processes plays a major role in completing the analogy. Appropriate mapping of the image gray scale intensity into a flow field creates conditions favorable for the formation or enhancement of field discontinuities. For example, a shock wave reflecting from a wall or a contact surface can increase in strength, or two colliding flow streams will produce a contact surface that will become stronger in time. If we have a numerical technique to resolve these discontinuities accurately, then with successive numerical integration of the flow field, the discontinuities will sharpen as the solution evolves in time. Then by inverse mapping of the flow field to the image gray scale field, we can reconstruct an enhanced image.

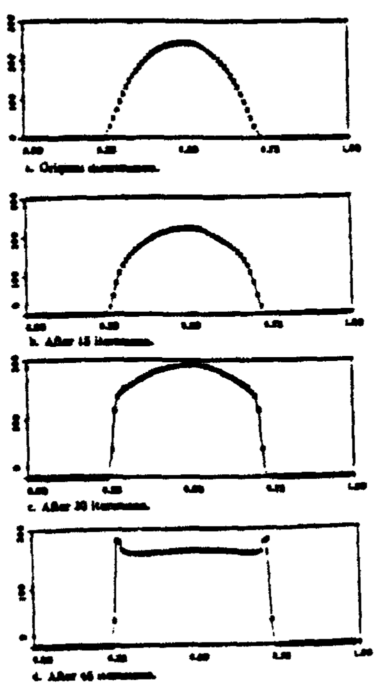


Figure 3.12.1 Edge enhancement for a sinusoidal distribution without noise.

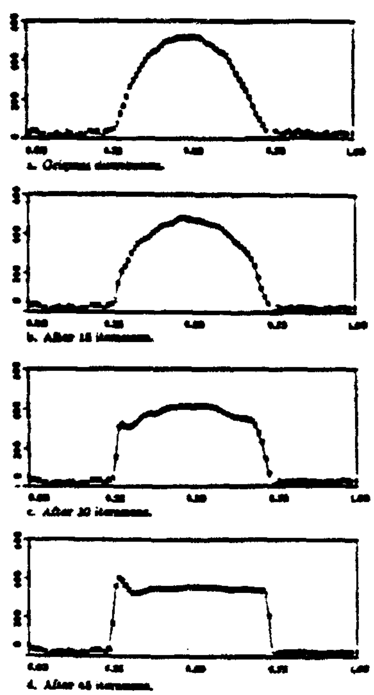


Figure 3.12.2 Edge enhancement for a sinusoidal distribution with 10% intensity random noise.

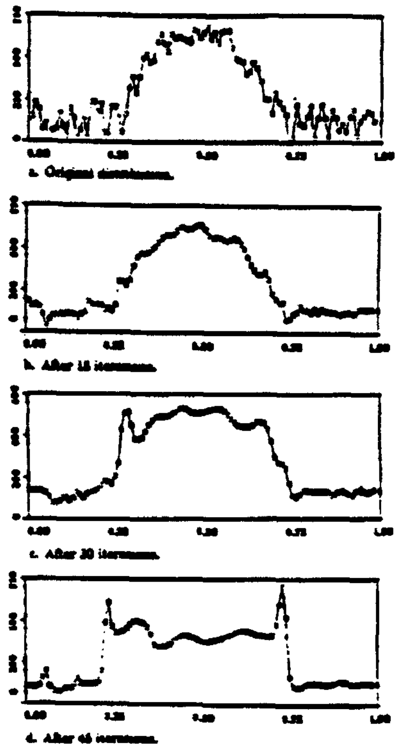


Fig. 3.12.3 Edge enhancement for a sinusoidal distribution with 50% intensity random noise.

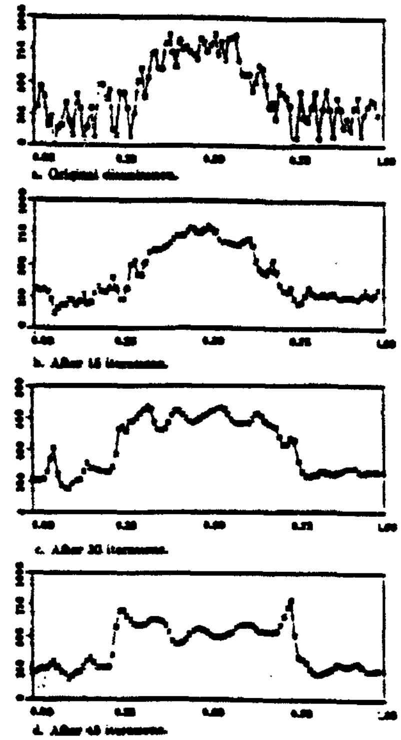


Fig. 3.12.4 Edge enhancement for a sinusoidal distribution with 100% intensity random noise.

Applications have been made to two-dimensional images derived from satellite reconnaissance and gamma-ray medical diagnostics (see Appendix C). Note that the images shown there are distorted by the xerographic process used to reproduce these illustrations, which also act as a nonlinear filter but is not tuned to these images.

Analogous extensions of nonlinear CFD techniques can be used for image compression.

3.13 DETONATION IN A MULTIPHASE MEDIUM

In this study the main subjects were the initiation, propagation, and structure of detonations occurring when combustible particles are intentionally or unintentionally dispersed into the air. Formation of this potentially explosive dust environment and the properties of its detonation are of significant practical interest in view of its destructive or creative effects. Previous experimental and theoretical studies of these phenomena addressed only homogeneous particle/oxidizer mixtures. However, intentional or accidental processes of the explosive dust dispersion always lead to inhomogeneous particle density distribution.

On the other hand, some industrial methods of explosive forming rely on detonation of explosive powder. This powder can be deposited as a thin layer over ··:e surface area of the forming metal, with a residual concentration in the vicinity of the la . r.

When the detonation wave is generated in a homogeneous mixture by " ct initiation," it starts with a strong blast wave from the initiating charge. As the blast .ve decays, combustion of the reactive mixture behind its shock front starts to have a .rger role in support of the shock wave motion. When the initial explosion energy e :eeds some critical value, transition to steady state detonation occurs. In explosiv. dust mixtures with a nonuniform particle density, the initiation dynamics is significantly more complicated. The critical initiation energy sufficient for one of the explosive particle density regions is not necessarily adequate for other regions. We have demonstrated that the phenomenology of these interactions is distinctly different from the classical studies of multilayer detonations in gases. This is primarily because the energy content of adjacent layers in a typical multigas layer experiment varies by a factor of two or four, whereas the energy content in explosive dust/air mixtures can vary by several orders of magnitude.

At present the physics of the energy release mechanisms in solid particles/air mixtures is not clearly understood. This can be attributed to the obvious difficulties of making a direct non-obtrusive measurement in the optically thick environment typical for this system. The chemical processes of single-particle combustion, which mainly occur in the gaseous phase, are significantly faster than the physical processes of particle gasification or disintegration. Thus, in the multiphase mixtures, the rate of energy release is mostly determined by physics of particle disintegration. It is very difficult to describe the details of particle disintegration in the complex environment prevalent behind the shock or detonation wave. Fortunately, in most cases of multiphase detonation, only the main features of the particle disintegration dynamics need to be captured to describe the phenomena.

In this work we considered solid particles consisting of explosive material. Two-dimensional simulations were done for the system of low particle density concentration clouds and ground layers formed by high concentrations of the RDX powder. We examined three cases of ground layer density distribution: a fourth power distribution with 12 mm above ground with a maximum density on the ground of 800 kg/m^3; a uniform 25-mm layer with a density of 100 kg/m^3; and a 12-mm uniform layer with a density of 250 kg/m^3. In all these cases, the weight of the condensed phase per unit area was the same, which allowed examination of the effects of the particle density distribution on detonation wave parameters.

Figure 3.13.1 shows a setup for a typical two-dimensional simulation. Here the computational domain is 25cm $\times$ 25cm. The explosive powder density is distributed according to the 4th power law of the vertical distance, starting from the ground where the density is 800 kg/m^3, and rising to 1.2cm, where the density is 0.75 kg/m^3. From this point to 25cm height, the density is constant and equal to 0.75kg/m^3. The density distribution is uniform in the x direction.

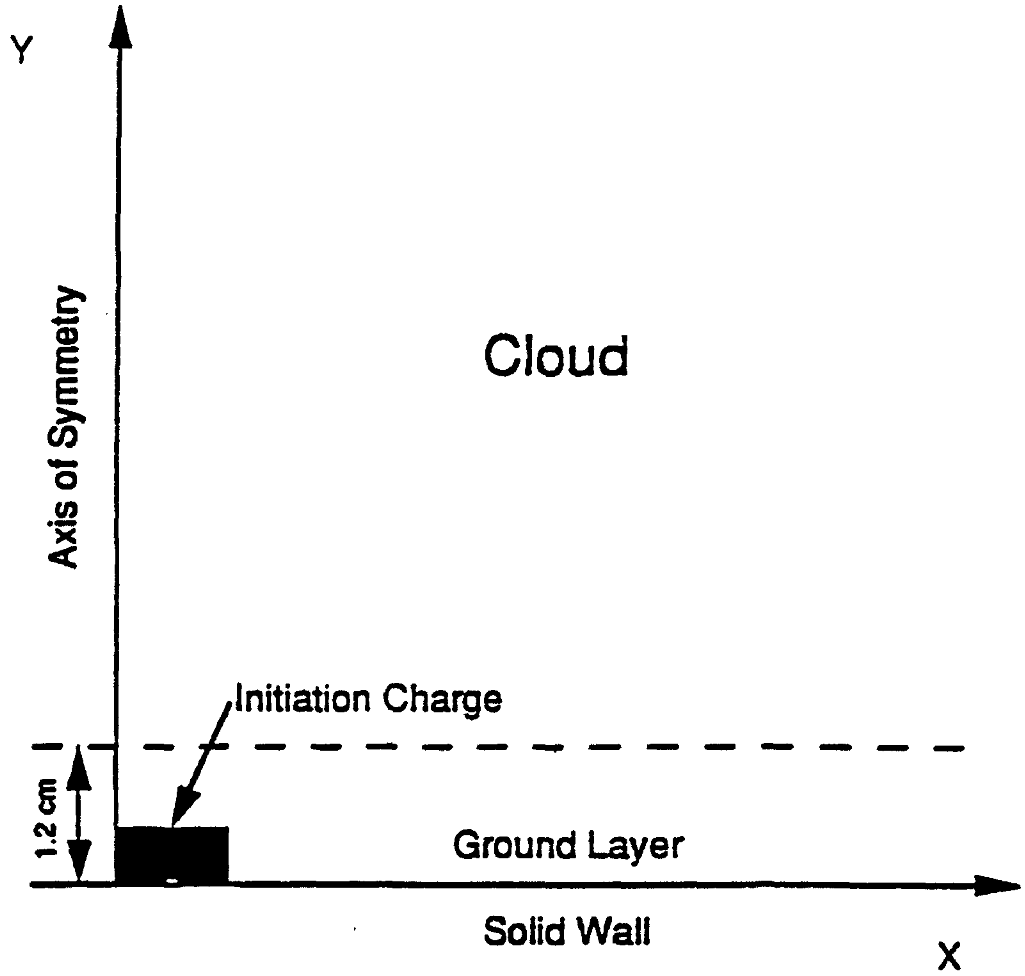


Figure 3.13.1 Computational domain and boundary conditions.

In all three cases, the detonation wave in the cloud in the computational domain was significantly overdriven and did not play an important role. We estimated that the self-sustained regime of the detonation wave in the cloud for the examined cloud concentrations can occur only at the distances of 2-3m above ground. At the same time, the particle density distribution in the layer determines the dynamics of the detonation wave as well as the pressure on the ground.

In all three two-dimensional simulations, we observed a very distinctive shape of the detonation wave front in the vicinity of the layer. In this area, the overdriven detonation in the cloud is preceding the detonation wave in the ground layer. This feature of the detonation front can be explained by the fact that the energy released in the ground layer detonation wave produces a faster propagating shock wave in the dilute cloud than in the ground layer which is heavily loaded with solid particles. However, these structures were not observed experimentally, and more studies are needed to examine their parameters.

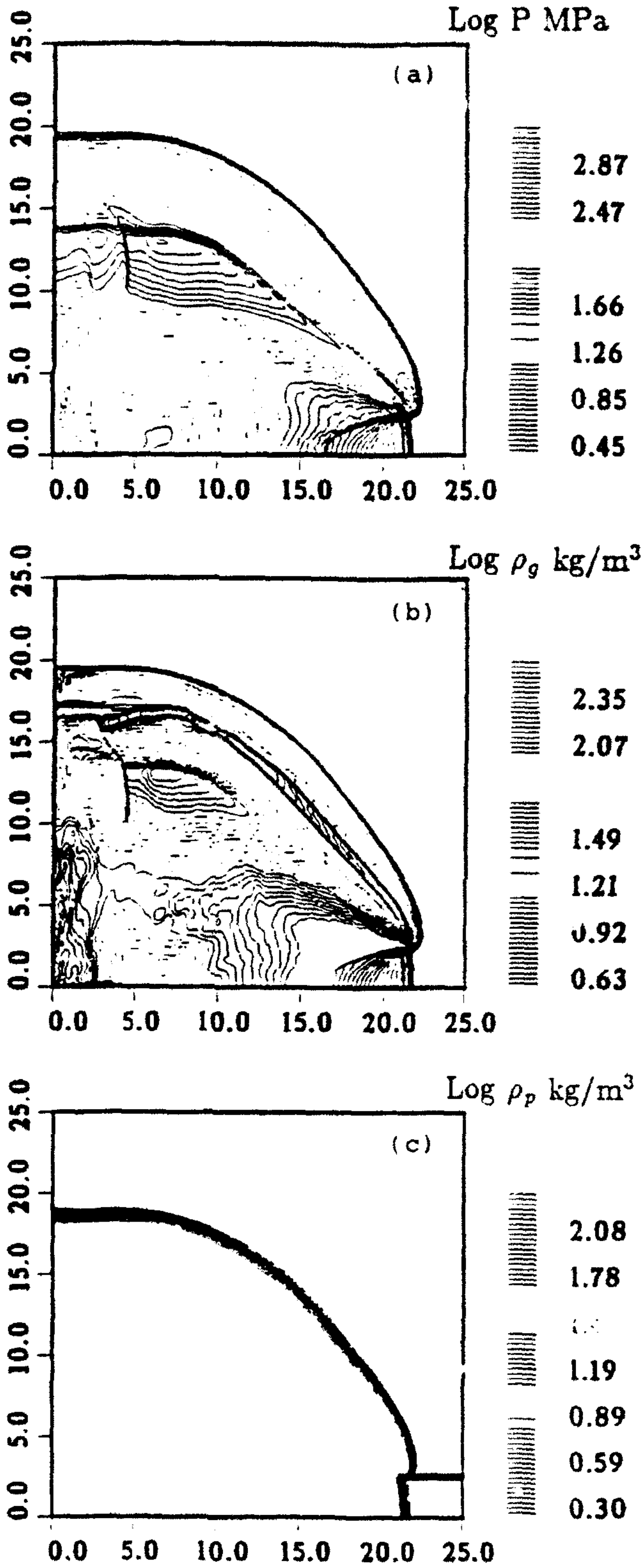


Figure 3.13.2 Explosive initially localized in 2.5-cm layer at constant density of 100 kg/m^3. Density in the cloud is 0.75 kg/m^3. (a), (b), and (c) are gas pressure, gas density, and particle density at 66 μsec, respectively.

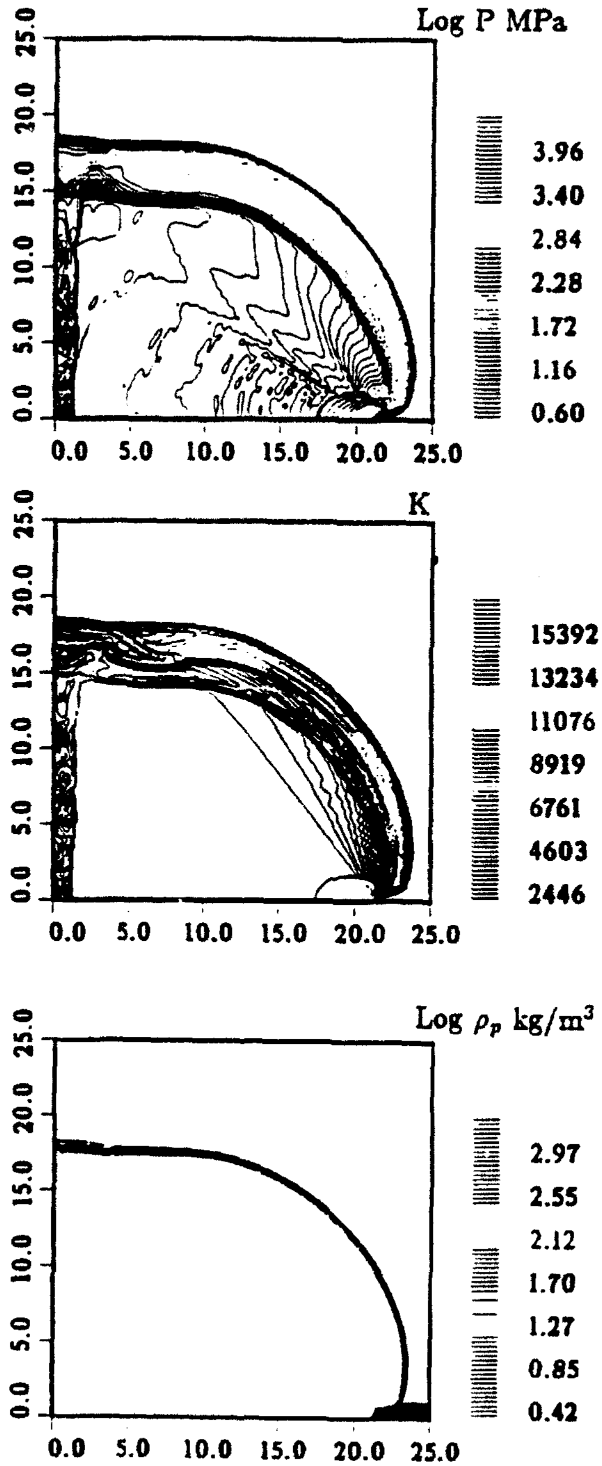


Figure 3.13.3 Particle density distributed in layer in accordance with the fourth power of height. Gas pressure, temperature, and particle density at 55 μsec, respectively.

4. CONCLUSIONS

The AUGUST-2D and AUGUST-3D adaptive unstructured CFD simulation codes, developed under SAIC's UUGM (through a contract form ARPA's Applied and Computational Mathematics Program) program have been tested through the set of standard CFD benchmark test cases and have been applied to a wide range of realistic problems for a variety of end-users. In most cases where these codes have been applied, significant improvements in accuracy, resolution, and ease of use have been noted. Use of the Second Order Godunov flow solver algorithm has provided a robust capability to treat low Mach number subsonic-to high Mach number hypersonic flow problems within one simulation code without the necessity of tuning the flow solver via adjustable parameters. In addition, the extension of the AUGUST family of codes to treat multiphase, multicomponent reactive flow phenomena provides the capability, for the first time, of simulating a wide variety of physically interesting and challenging problems that are rich in physics-chemical phenomena. The range of these problems includes: 1) full 3D flows about complex aircraft in all flight regimes (except rarefied flows), 2) shock-body interactions, 3) chemically reacting flows typical in combustion problems, and 4) detonation phenomena found in explosives, shock tubes, and specific applications to such devices as the pulsed detonation engine.

SAIC's UUGM program has resulted in over 20 publications in various stages of preparation, and numerous presentations at U.S. and international technical meetings, conferences, and workshops. The AUGUST family of simulation codes is presently being applied to several current materials development and synthesis areas of research. In particular, the ability of the AUGUST codes to capture the complex geometry of material synthesis reactor configurations, resolve the complex flow patterns, and treat the complex physics and chemistry of the synthesis process provides a simulation and modeling tool that is useful for design of such process reactors, analyse and evaluate experimental results, and (depending on successful benchmarking) provide a process control tool based on validated models. SAIC intends to exploit this capability in future programs.

SAIC's Applied Physics Operation, Hydrodynamic Modeling Division staff members performed the work under the DARPA UUGM program. Dr. Shmuel Eidelman and Dr. William Grossmann were co-program managers. Important contributions were made by Drs. Itzhak Lottati, Xiaolong Yang, Marty Fritts, Adam Drobot, Ahron Friedman, and Michael Kress. SAIC's UUGM team would like to acknowledge the support and interest of Dr. James Crowley (ARPA ACMP program manager), Drs. Lois Auslander and Helena Wisniewski (previously DARPA ACMP program managers), and Dr. Arje Nachman (AFOSR) who served as the ARPA agent for the UUGM program.

REFERENCES

1. T.J. Baker and A. Jameson, "A Novel Finite Element Method for the Calculation of Inviscid Flow Over a Complete Aircraft," Sixth International Symposium on Finite Element Methods in Flow Problems, Antibes, France (1986).

2. T.J. Baker, "Developments and Trends in Three-Dimensional Mesh Generations," Transonic Symposium held at NASA Langley Research Center, Virginia (1988).

3. R. Lohner, "Generation of Three-Dimensional Unstructured Grids by the Advanced-Front Method," AIAA 26th Aerospace Sciences Meeting, Reno, AIAA Paper 88-0515, January 1988.

4. R. Lohner and K. Morgan, "Improved Adaptive Refinement Strategies for Finite Element Aerodynamic Computations," AIAA 29th Aerospace Sciences Meeting, Reno, AIAA Paper 86-0499, January 1986.

5. A. Jameson, T.J. Baker, and N.P. Weatherill, "Calculation of Inviscid Transonic Flow Over a Complete Aircraft," AIAA 24th Aerospace Sciences Meeting, Reno, NV, AIAA Paper 86-0103, January 1986.

6. R. Lohner, "Adaptive Remeshing for Transient Problems, Comp. Meth. Appl. Mech. Eng. 75 195-214 (1989).

7. J. Peraire, M. Vahdati, K. Morgan, and O.C. Zienkiewicz, "Adaptive Remeshing for Compressible Flow Computations," J. Comp. Phys. 72, 449-466 (1987).

8. D. Mavriplis, "Accurate Multigrid Solution of the Euler Equations on Unstructured and Adaptive Meshes," AIAA 88-3707 (1988).

9. I. Lottati, S. Eidelman, and A., Drobot, "A Fast Unstructured Grid Second Order Godunov Solver (FUGGS)," 28th Aerospace Sciences Meeting, AIAA-90-0699, Reno, NV (1990).

10. I. Lottati, S. Eidelman, and A. Drobot, "Solution of Euler's Equations on Adaptive Grids Using a Fast Unstructured Grid Second Order Godunov Solver," Proceeding of the Free Lagrange Conference, Jackson Lake, WY, June 1990.

11. I. Lottati and S. Eidelman, "Second Order Godunov Solver on Adaptive Unstructured Grids," Proceeding of the 4th International Symposium on Computational Fluid Dynamics, Davis, CA, September 1991.

12. S. Eidelman, P. Collela, and R.P. Shreeve, "Application of the Godunov Method and Its Second Order Extension to Cascade Flow Modeling," AIAA Journal 22, 10 (1984).

13. B. van Leer, "Towards the Ultimate Conservative Difference Scheme, V.A. Second Order Sequel to Godunov's Method," J. Comp. Phys. 32, 101-136 (1979).

14. P. Collela and P. Woodward, "The Piecewise Parabolic Method (PPM) for Gas-dynamic Simulations," J. Comp. Phys. 54, 174-201 (1984).

15. J.F. Thompson, "Grid Generation Techniques in Computational Fluid Dynamics," AIAA Tour., Vol. 22, No. 11, pp. 1505-1523, November, 1984.

16. M.S. Shepherd and M.K. Georger, "Automatic Three-Dimensional Mesh Generation by the Finite Octree Method," Intern. J. Num. Meth. Eng., Vol. 32, pp. 709-749, (1991).

17. H.M. Glaz, P. Colella, L.I. Glass, and R.L. Deschambault, "A Detailed Numerical, Graphical and Experimental Study of Oblique Shock Wave Reflections," DNA-TK-86-365, 1986.

18. I.I. Glass and D.L. Zhang, "Interferometric Investigation of the Diffraction of Planar Shock Waves Over a Half-Diamond Cylinder in Air," UTIAS Report No. 322, March 1988.

19. H. Hornung, "Regular and Mach Reflection of Shock Waves," Annual Review of Fluid Mechanics, Vol. 18, pp. 33-58, 1986.

PUBLICATIONS UNDER THE UUGM PROJECT

1. S. Eidelman, W. Grossmann, and I. Lottati, "A Review of Propulsion Applications of the Pulsed Detonation Engine Concept," AIAA 89-2446, AIAA/ASME/SAE/ASEE 25th Joint Propulsion Conf., Monterey, CA, July 1989.

2. S. Eidelman, W. Grossmann, and I. Lottati, "Computational Analysis of Pulsed Detonation Engines and Applications," AIAA 90-0460, 28th Aerospace Sciences Meeting, Reno, NV, January 1990.

3. I. Lottati, S. Eidelman, and A. Drobot, "A Fast Unstructured Grid Second Order Godunov Solver (FUGGS)", AIAA 90-0699, 28th Aerospace Sciences Meeting, Reno, NV, January 1990.

4. S. Eidelman and I. Lottati, "Reflection of the Triple Point of the Mach Reflection in a Planar and Axisymmetric Converging Channels," 9th Mach Reflection Symposium, Freiburg, Germany, June 1990.

5. I. Lottati, S. Eidelman, and A. Drobot, "Solution of Euler's Equations on Adaptive Grids Using A Fast Unstructured Grid Second Order Godunov Solver (FUGGS)," in H.E. Trease, M.J. Fritts, and W.P. Crowley (Eds.), Proceedings of the Next Free-Lagrange Conference, Jackson Lake, WY, June 1990 [*Advances in the Free-Lagrange Method*, Springer-Verlag, New York (1992)].

6. S. Eidelman, W. Grossmann, and I. Lottati, "Air-Breathing Pulsed Detonation Engine Concept; A Numerical Study," AIAA 90-2420, AIAA/SAE/ASME/ASEE 26th Joint Propulsion Conf., Orlando, FL, July 1990.

7. E. Hyman, K. Tsang, I. Lottati, A. Drobot, B. Lane, R. Post, and H. Sawin, "Plasma Enhanced Chemical Vapor Deposition Modeling," *Surface and Coatings Tech.* **49**, 387 (1991).

8. S. Eidelman, W. Grossmann, and A. Friedman, "Nonlinear Signal Processing Using Integration of Fluid Dynamics Equations," Applications of Digital Image Processing XIV, SPIE Vol. 1567, 1991.

9. S. Eidelman, W. Grossmann, and I. Lottati, "Review of Propulsion Applications and Numerical Simulations of the Pulsed Detonation Engine Concept," *J. Propulsion* 7, 857 (1991).

10. D.L. Book, S. Eidelman, I. Lottati, and X. Yang, "Numerical and Analytical Study of Transverse Supersonic Flow Over a Flat Cone," Shock Waves 1, 197, 1991.

11. S. Eidelman and X. Yang, "Detonation Wave Propagation in Variable Density Multi-phase Layers," AIAA 92-0346, 30th Aerospace Sciences Meeting, Reno, NV, January 1992.

12. S. Eidelman, I. Lottati, and W. Grossmann, "A Parametric Study of the Air-Breathing Pulsed Detonation Engine," AIAA 92-0392, 30th Aerospace Sciences Meeting & Exhibit, Reno, NV, January 1992.

13. I. Lottati and S. Eidelman, "A Second Order Godunov Scheme on a Spatial Adapted Triangular Grid," in U.S. Army Workshop on Adaptive Methods for Partial Differential Equations, Rensselaer, NJ, 1992.

14. I. Lottati and S. Eidelman, "Decomposition by Structured/unstructured Composite Grids for Efficient Integration in Domains with Complex Geometries," in *Adv. in Computer Methods for Partial Differential Equations VII*, R. Vichnevetsky, D. Knight, and G. Richter (Eds.), 1992.

15. X. Yang, S. Eidelman, and I. Lottati, "Two-Phase Compressible Flow Computation on Adaptive Unstructured Grid Using Upwind Schemes," in *Adv. in Computer Methods for Partial Differential Equations VII*, R. Vichnevetsky, D. Knight, and G. Richter (Eds.), 1992.

16. S. Eidelman and W. Grossmann, "Pulsed Detonation Engine Experimental and Theoretical Review," AIAA 92-3168, AIAA/SAE/ASME/ASEE 28th Joint Propulsion Conf. and Exhibit, Nashville, TN, July 1992.

17. S. Eidelman and A. Altshuler, "Synthesis of Nanoscale Materials Using Detonation of Solid Explosives," *1st Intern. Conf. on Nanostructured Materials*, Cancun, Mexico, September 1992.

18. S. Eidelman and X. Yang, "Detonation Wave Propagation in Combustible Mixtures with Variable Particle Density Distributions," AIAA J. **31**, 228, 1993.

19. S. Eidelman and X. Yang, "Detonation Wave Propagation in Combustible Particle/Air Mixture with Variable Particle Density Distributions," Combust. Sci. and Tech. **89**, 201, 1993.

20. X. Yang, S. Eidelman, and I. Lottati, "Computation of Shock Wave Reflection and Diffraction Over a Semicircular Cylinder in a Dusty Gas," AIAA 93-2940, 24th Fluid Dynamics Conf., Orlando, FL, July 1993.

21 I. Lottati and S. Eidelman, "Acoustic Wave Focusing in an Ellipsoidal Reflector for Extracorporeal Shock-wave Lithotripsy," AIAA 93-3089, 24th Fluid Dynamics Conf., Orlando, FL, July 1993.

APPENDIX A

CODE DESCRIPTION

APPENDIX A
CODE DESCRIPTION

A.1 AUGUST (2D)

The subroutines in the AUGUST code are organized here as they appear in the listing in Appendix B. A brief description indicates the function performed by each subroutine.

TABLE A.1

LIST OF SUBROUTINES

1. **MAIN**	Governing program for AUGUST. Reads input files and sets the mode for the computation.
2. **HYDRFL**	Computes the fluxes at interfaces by applying the Godunov algorithm to solve the Riemann problem across the interface.
3. **HYDRMN**	Controls the computation. The integration of the fluxes and update of the physical variables, adaptation of the grid and writing to output files are performed in this subroutine.
4. **GEOMTR**	Calculates the geometrical quantities not provided by the input data file but needed for the computational algorithm. GEOMTR is only used once for starting a new simulation.
5. **UPDATE**	Reads the input file for a new simulation and calls GEOMTR to update the geometrical variables needed to perform the computation.

6. **UPGRAD**	Called if a restart run is performed. Will read the appropriate file written at the end of the previous run.
7. **GRADNT**	Computes the gradient of the physical variables to improve the prediction of those variables for the two sides of the interface. The gradients are subjected to the monotonicity condition that limits the projected values, thus preventing new *maxima-minima from being caused* artificially by interpolation (IOPORD = 2). Calls FCHART in order to compute projected values at the half timestep associated with the local characteristics of the flow.
8. **GRDFLX**	Computes the gradient of the pressure and Mach number in each cell. This information is used as an error indicator for the adaptation needed in a steady state solution.
9. **FIRST**	The equivalent of GRADNT if run in a first order mode (IOPORD = 1). Using FIRST assumes that the physical variables are constant in each cell. Takes care of the boundary conditions if the interface is a boundary.
10. **FCHART**	Computes the projected values at a half timestep for the two sides of the interface based on the local characteristics of the flow. Called by GRADNT, it modifies the projected values for the two sides of the interface and assigns them to the correct location in memory. Takes care of the boundary conditions if the interface is a boundary.

11. **PRLCTN**	Determines particle cell location in the initial phase of tracing a group of particles.
12. **PRPTHC**	Advances the position of each particle, assuming that the particle has the flow velocity of the cell. PRPTHC will find the cell location of the particle after it advances by the timestep of the computation.
13. **VERCEN**	Places an additional vertex at the center of a specified cell to refine the size of the cell by a factor of three.
14. **DISECT**	Places an additional vertex at the middle of a specified edge to refine the size of the two cells adjacent to the edge by a factor of two. This method of refinement is used only on the edges lying on the boundaries of the computational domain.
15. **DYNPTN**	Tests and flags the cells for specified refinement criteria. DYNPTN is called only if the parameter IOPADD = 1. Will start the refinement procedure by calling VERCEN and DISECT and will call DYYPTN for further refinement. This insures that the buffer zone ahead of the shock is resolved according to the specified area criteria (AREADD).
16. **DYYPTN**	Refines the cells flagged by DYNPTN by calling VERCEN and DISECT until the area of each flagged cell meets the area criteria specified by the parameter AREADD.

17. **INTPTN**	Refines the cells in the inlet region. Prepares the inlet region for the introduction of a shock wave. This initial refinement is essential to prevent additional refinement of the grid in the presence of a shock wave. It is called only if the parameters ICOND=0 and IOPTN= 2 (solution for transient phenomena).
18. **DELPTN**	Tests and flags the cells for the specified criteria for coarsening. DELPTN is called only if parameter IOPDEL = 1.
19. **RELAXY**	Relaxes the vertices of the cells that were created in the process of deleting a vertex.
20. **VERDEL**	Deletes a specified vertex.
21. **RECNC**	Tests two cells adjacent to the specified edge. Compares them to the two cells that can be created if this edge is flipped to pass between the other two vertices of the quadrilateral containing the original two cells. If the tests result in a better quality triangle, then RECNC will swap the edge.
22. **EOS**	Applies Gilmore equation of state to compute $\gamma = c_p/c_v$, giving the internal energy and density of the fluid in a cell. This option is controlled by the parameter IOPEOS = 1.
23. **LIFTDR**	A diagnostic to compute the lift, drag, and transfer momentum developed in the configuration. Takes into account all boundary edges that are specified as 5. It is controlled by the parameter IOPLFT = 1.

THE MAIN PROGRAM

All of the data input and initiation of a run (or a restart run) is performed in MAIN. The actual simulation is controlled by HYDRMN, which is called from MAIN. At the completion of a run, control is returned to MAIN and a successful termination prints the message STOP 777.

MAIN contains one name list (file no. 2) and requires an input file that contains tne grid data description (file no. 16). The data organization for the grid file is described in Appendix A. There are five files that should be included: CINT00.H, CMSH00.H, CPHS10.H, CPHS20.H, CHYD00.H.

NAMELIST/DATA	ICOND	ICONP	ITRIGR	IOPTN
	XMCHIN	RIN	PIN	ALFA
	HRGG	IHRN	NTIME	MDUMP
	NDUMP	KDUMP	IOSPCL	IOPLFT
	IOPRCN	IOPORD	IOPBYN	IAXSYM
	IOPEOS	MPRTCL	IOPINT	IOPADD
	IOPDEL	AREADD	AREDEL	IWINDW
	ISTATC			

VARIABLE	PURPOSE
ICOND	= 0 READ INPUT GRID FOR A NEW SIMULATION = 1 READ THE GRID FROM PREVIOUS RUN

ICOND = 0: MAIN will read the initial grid definition stored in file number 16. The current setting is to read the inpu: :le as provided by Smart, a two dimensional triangul... grid generator that runs interactively on the Macintosh p·.rsonal computer.

MAIN will call UPDATE, which will call C:EOMTR. GEOMTR will compute essential geometrical param:eters that are not provided by file 16. All geometrical information is dumped into output files (8 and 88) so that ICOND=0 is used only once at the beginning of a new simulation.

ICOND = 1: MAIN will call UPGRAD, which will call one of the output files (8 or 88) written by the previous run. This will load the geometrical definition of the grid (either 8 or 88---they are identical). Writing identical files provides a backup in the event that the job terminates for lack of time while in the process of writing to one of those output files.

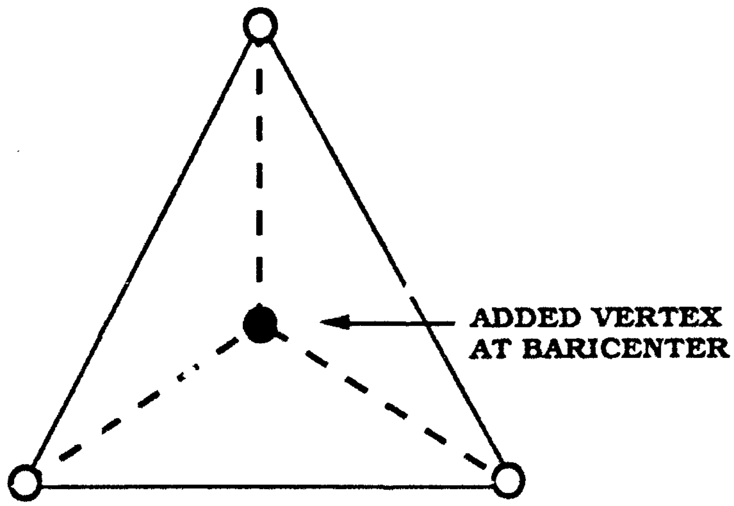
VARIABLE	PURPOSE
ICONP	= 0 PRIMITIVE VARIABLES INITIALIZED = 1 VARIABLES READ FROM PREVIOUS RUN

ICONP = 0: Initialize the primitive variables in computational domain with an initial value specified by the user. The two options set by the code are controlled by IOPTN.

ICONP = 1: The flow field condition reads in files 8 or 88 and provides a followup run set from the previous run.

VARIABLE	PURPOSE
ITRIGR	= 0 USING THE INPUT GRID AS THE INITIAL GRID = 1 THE INPUT GRID TRIPLED BY ADDING AN EXTRA VERTEX IN EACH TRIANGLE

ITRIGR = 1: The original grid cells will be tripled by adding an extra vertex in the baricenter of each triangle. This option can be triggered at the beginning of a simulation only (ICOND = 0).



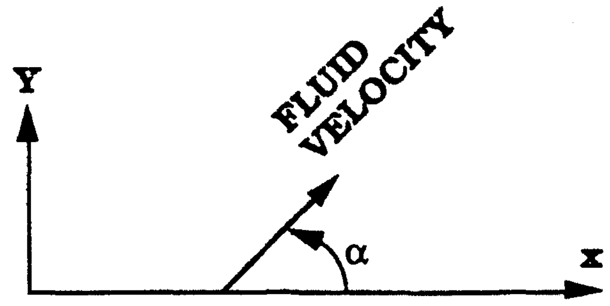
VARIABLE	PURPOSE
IOPTN	= 1 SOLUTION FOR STEADY STATE = 2 SOLUTION FOR TRANSIENT PHENOMENA

There are two choices available to set the initial condition of the problem.

IOPTN = 1: Assign the conditions at the inlet to the computational domain. This is the fastest way to get a steady state solution for the conditions specified at the inlet. In this option, PIN (pressure), RIN (density) and XMCHIN (Mach number) are assigned to the pressure density and velocity (the speed of sound is computed in the code) and imposed at the inlet boundaries.

IOPTN = 2: Used if a shock wave is to be simulated moving from the inlet (edge boundary 8) to the outlet (edge boundary 7). For this setting, specify PIN (ambient pressure in the chamber), RIN (ambient density in the chamber) and XMCHIN (upstream Mach number). The code will use the normal shock wave relations for an adiabatic flow of a completely perfect fluid to compute the static-pressure ratio across the shock P_2/P_1 and the density ratio ρ_2/ρ_1, and the ratio of the Mach number across the shock M_2/M_1. These computed quantities are applied to set correctly the condition on the pressure density and velocity at the inlet boundary.

VARIABLE	PURPOSE
ALPHA	THE DIRECTION OF INFLOW IN DEGREES RELATIVE TO A RIGHT HAND COORDINATE SYSTEM. ALPHA = 0 MEANS FLOW FROM LEFT TO RIGHT.



The velocity computed by the code according to the input data provided by the user is split (projected) in the X and Y directions by using α.

VARIABLE	PURPOSE
HRGG	INITIAL γ IN THE EQUATION OF STATE. THE CODE RUNS USING THE IDEAL EQUATION OF STATE AS A BASELINE AND SHOULD BE MODIFIED IF SOMETHING ELSE IS DESIRED. IOPEOS=1 WILL TRIGGER THE USE OF GILMORE EQUATION OF STATE.

VARIABLE	PURPOSE
IHRN	NUMBER OF ITERATIONS IN THE RIEMANN SOLVER TO FIND THE DIAPHRAGM SOLUTION. (THREE TO FOUR SHOULD BE USED AND INCREASED ONLY FOR A VERY HIGH MACH NUMBER CASES.)

VARIABLE	PURPOSE
NTIME	NUMBER OF REPEATS FOR THE INTEGRATION/ REFINEMENT/COARSENING SEQUENCE. AN OUTPUT DUMP IS DONE FOR EVERY SEQUENCE REPEAT.

VARIABLE	PURPOSE
MDUMP	NUMBER OF OUTERMOST LOOP ITERATIONS IN THE CALCULATION WHERE COARSENING OF THE GRID IS PERFORMED EVERY SEQUENCE REPEAT.

VARIABLE	PURPOSE
NDUMP	NUMBER OF OUTER LOOP ITERATIONS IN THE CALCULATION WHERE REFINING IS DONE FOR EVERY SEQUENCE REPEAT WITHOUT COARSENING.

VARIABLE	PURPOSE
KDUMP	NUMBER OF ITERATIONS PERFORMED WITH NO REFINEMENT OR COARSENING. THE INNER LOOP OF THE CALCULATION. IF KDUMP = 0, KDUMP WILL BE SET BY THE CODE AUTOMATI-CALLY ACCORDING TO THE SETTING OF THE VARIABLE AREADD.

O NTIME - DUMPING DATA
O MDUMP - COARSENING
O NDUMP - REFINEMENT
O KDUMP - INTEGRATION

O CLOSING INNER LOOP
O CLOSING OUTER LOOP
O CLOSING OUTERMOST LOOP
O CLOSING DUMPING LOOP

VARIABLE	PURPOSE
IOSPCL	= 0 NOT USING REDEFINITION OF POINTS ON THE BOUNDARY = 1 USING REDEFINITION OF POINTS ON THE BOUNDARY

IOSPCL = 1: Modifies the definition of points along the boundary according to a presetting in the code. The setting currently will redefine the points along the edge boundary 5 to exactly match NACA0012 airfoil shape. This is done to redefine points on a boundary that has an analytical definition of points, but where these points have been dislocated by a refining procedure.

VARIABLE	PURPOSE
IOPLFT	= 0 THE COMPUTATION OF LIFT DRAG AND MOMENT TURNED OFF = 1 THE COMPUTATION OF LIFT DRAG AND MOMENT TURNED ON

Set IOPLFT = 1 if integral quantities need to be computed. The current setting will calculate the lift, drag and moment on edge boundary 5.

VARIABLE	PURPOSE
IOPRCN	= 0 A GLOBAL SWAPPING (RECONNECTION) PROCEDURE IS OFF = 1 A GLOBAL SWAPPING (RECONNECTION) PROCEDURE IS ON

This swapping is done by calling subroutine RECNC. It is used only in a new simulation (ICOND = 0).

VARIABLE	PURPOSE
IOPORD	= 1 THE CODE WILL RUN FIRST ORDER GODUNOV METHOD = 2 THE CODE WILL RUN SECOND ORDER GODUNOV METHOD

IOPORD = 1 Subroutine FIRST is called.

IOPORD = 2 Subroutine GRADNT is called.

VARIABLE	PURPOSE
IOP[illegible]YN	= 0 NO BUOYANCY EFFECTS ARE COMPUTED = 1 BUOYANCY EFFECTS IN THE X DIRECTION ARE COMPUTED = 2 BUOYANCY EFFECTS IN THE Y DIRECTION ARE COMPUTED

The buoyancy effect applies the gravity acceleration as g = 9.81.

VARIABLE	PURPOSE
IAXSYM	= 0 THE CODE WILL RUN IN A PURE TWO DIMENSIONAL MODEL = 1 THE CODE WILL RUN IN AN AXISYMMETRICAL MODE (X AS THE AXIS OF SYMMETRY) = 2 THE CODE WILL RUN IN AN AXISYMMETRICAL MODE (Y AS THE AXIS OF SYMMETRY)

VARIABLE	PURPOSE
IOPEOS	= 0 THE CODE WILL RUN WITH CONSTANT γ = 1 THE CODE WILL RUN WITH VARIABLE γ USING THE EQUATION OF STATE FOR AIR

IOPEOS = 0: The initial γ is not changed and is kept constant across the computational domain at all times (with value set by HRGG).

IOPEOS = 1: The γ of each cell will be modified according to local internal energy and density. Thus, if IOPEOS = 1, the actual pressure and density should be input (in the appropriate dimension). Otherwise (IOPEOS=0), a normalized pressure and density of unity can be used for simulation.

VARIABLE	PURPOSE
MPRTCL	= 0 NO PARTICLE TRACING = 1 THE CODE WILL TRACE PARTICLES

MPRTCL = 1: The ability to trace particles will be turned on. Initially PRLCTN is called to identify the cell location of each particle. For each time step, PRPTHC will be called to update the cell location of each particle if it is relocated, assuming the particle moves at the same velocity as the fluid.

The initial location of the particles is defined in MAIN.

VARIABLE	PURPOSE
IOPINT	= 0 DOES NOT PREPARE A BUFFER ZONE. = 1 INITIALLY PREPARE A BUFFER ZONE AHEAD OF EDGE BOUNDARY 8

For simulating transient phenomena, the refining of the grid is done in the region ahead of the shock. In this way, we avoid interpolating in a region where large gradients reside. IOPINT = 1 will refine the region of the inlet flow to prepare a buffer zone (edge boundary 8). If refining is needed in another region, subroutine INTPTN should be modified accordingly.

VARIABLE	PURPOSE
IOPADD	= 0 THE REFINEMENT PROCEDURE IS TURNED OFF = 1 THE REFINEMENT PROCEDURE IS TURNED ON

VARIABLE	PURPOSE
IOPDEL	= 0 THE COARSENING PROCEDURE IS TURNED OFF = 1 THE COARSENING PROCEDURE IS TURNED ON

VARIABLE	PURPOSE
AREADD	SPECIFIES THE MINIMUM AREA VALUE THAT A TRIANGLE SHOULD HAVE AFTER REFINEMENT. SPECIFIED AS A FRACTION OF THE AVERAGE TRIANGLE AREA OF THE INITIAL GRID. THIS REFERENCE AREA IS KEPT CONSTANT THROUGH THE WHOLE SIMULATION.

VARIABLE	PURPOSE
AREDEL	SPECIFIES THE MAXIMUM VALUE THAT A TRIANGLE SHOULD HAVE AFTER COARSENING DEFINED AS A FRACTION OF THE REFERENCE AREA.

VARIABLE	PURPOSE
IWINDW	= 0 NO RESTRICTION ON THE REGION FOR REFINING THE GRID = 1 SETTING A WINDOW FOR REFINING THE GRID

IWINDW = 1: The user can specify a region in which the refinement process will take place. Otherwise, the refinement takes place everywhere in the computational domain.

VARIABLE	PURPOSE
ISTATC	= 0 THE ADAPTATION WILL BE DONE ON A MOVING WAVE = 1 THE ADAPTATION WILL BE DONE ON A STEADY STATE CONDITION

Because the criteria for refinement in the presence of a static shock are not suited to treating a moving shock, the code sets different error indicators for adapting the grid for the two cases.

ISTATC = 0: The energy and density net fluxes across each cell are tested for sensing the level of activity. This method is a very good error indicator for sensing transient phenomena as traveling shocks.

ISTATC = 1: The pressure and Mach gradients in each cell are tested for sensing steady state shocks.

The gradient of density is always tested as a third criteria for making sure that static shocks are not ignored in computing a transient flow.

HYDRFL

Computes the fluxes across interfaces when the conditions for both sides are given. The fluxes are computed assuming a shock solution at a broken diaphragm simulated by the presence of the interface. The conditions existing on the two sides of the diaphragm will define the condition of the flow at the diaphragm location. These conditions are computed by solving the Riemann problem using the Godunov algorithm. The condition at the diaphragm defines the flux of energy, mass, and momentum passing across the interface. The Euler conservation law is applied to conserve energy, mass, and momentum crossing interfaces from one cell to the other.

Quantity	Side 1	Diaphragm (Interface)	Side 2
Density	ρ_1	ρ	ρ_2
Pressure	P_1	P	P_2
Velocity Perpendicular to Interface	u_1	u	u_2
Velocity Parallel to Interface	v_1	v	v_2

HYDRMN

Controls the code and the iteration loops. It calls HYDRFL to find the interface fluxes. These fluxes are integrated to update the physical variables in each cell. If adaptation of the grid is required, HYDRMN will set the criteria for controlling the adaptation of the grid. The refining (DYNPTN, DYYPTN) and coarsening (DELPTN) of the grid is invoked by HYDRMN. HYDRMN also controls the output by writing the necessary information on files for post processing data and for restarting the AUGUST code at a later time. It also manages print file diagnostics.

GEOMTR

Calculates geometrical variables that are not supplied by the input data and are needed to run the code. For example, it will compute:

1) Area of the cells;
2) Length of the edges;
3) Unit vector perpendicular to the edge. (For boundary edges, this unit vector is direct from the computational domain outward);
4) Unit vector directed from the baricenter of the left cell to the baricenter of the right cell. For boundary edges, the unit vector is perpendicular to the edge (from left cell outward).

The code will change the direction of the boundary edges so that all are arranged counterclockwise and the associated computational cell is always on the left side. GEOMTR is called once in the beginning of a new simulation.

UPDATE

Called in the beginning of a new simulation for setting geometrical variables not provided by the input data. (It calls GEOMTR.)

UPGRAD

Called if the run is a restart. *UPGRAD* will read the appropriate file (either 8 or 88) dumped by the previous run.

GRADNT

Compute the gradients of the physical variables in each cell. These computed gradients, along with the physical values at the baricenters, are applied using linear interpolation to predict the values on the interface.

The computed gradients are subjected to the monotonicity condition, ensuring that the projected values are bounded by the value of each quatity in the three adjacent cells, and to make sure that no new maxima or minima occurs. The projection of quantities to the interface improves the results from the code and provides second order accuracy in space.

GRADNT calls FCHART, which computes the projected values at the interfaces at the half timestep level according to the local characteristics of the flow in each cell bordering the interface cell. The assignment of values at the *two sides of each interface* is done at the end of FCHART. This same loop will also impose the boundary conditions for the interfaces at the boundaries of the computational domain.

GRDFLX

Computes the gradient of the Mach value and pressure gradient in each cell. These gradients are applied if the adaptation is done on a steady state converged solution. These variables, in addition to the computed density gradient, provide the criteria for adaptation if it is necessary to refine the grid for steady state problems.

FIRST

Assigns flow quantities to each side of an edge. These are based on the values at the baricenter of the triangles on either side of the edge. FIRST uses a first order approximation to find the values at the edge.

The user can specify FIRST or GRADNT by choosing 1 or 2 for the parameter IOPORD.

FCHART

Called by GRADNT to compute the values projected at the interfaces at the half timestep. These calculations are done by applying the local velocity characteristics in each cell. This projection in time improves the results and makes the code second order accurate in time.

PRLCTN

Identifies the initial cell location of each particle. Called once after specifying the starting location of each particle to be traced.

PRPTHC

Advances the particle position by the marching timestep. It finds the new cell location if a particle crosses an interface. The assumption is that the particles move at the fluid velocity.

VERCEN

Introduces a new vertex at the baricenter of the designated cell during the refinement process.

DISECT

Introduces a new vertex at the middle of a designated edge.

DYNPTN

Tests the cells according to the refining criteria and flags each cell which requires refinement. The flagged cells are refined in DYYPTN. The refinement is subjected to geometrical constraints on the cell shape to retain a high better quality refined grid.

The user can specify a window in the computational domain for refinement. The parameter to trigger this option is IWINDW = 1. For specifying the actual window, it may be necessary for the user to alter this subroutine and provide a definition of the geometrical area to be refined.

DYYPTN

Traces the cells that are flagged for refinement by DYNPTN. It subdivides them until each one of the refined cells meets the area refinement criteria of AREADD. Because each loop of refinement is restricted to a one-third reduction in cell area (calling VERCEN), DYYPTN will perform the necessary number of loops to meet the area reduction specified for refinement. AREADD is a fraction of the average area of the initial grid. This reference area is kept constant and fixes the minimum resolution in the simulation domain.

INTPTN

Performs the initial refinement of the grid before the initialization. The assumption is that a shock wave is introduced through the inlet boundary. Consequently, INTPTN will test for the inflow boundary interface and will refine the appropriate cells. (Note: It is not recommended that the code automatically refine the grid in the inlet region in the presence of a shock wave. If a shock wave is not introduced through the inlet, INTPTN should be modified to accommodate the change of the initial condition.)

DELPTN

Tests the cells according to coarsening criteria and flags them. Each triangle is tested to determine which vertex of the triangle is most appropriate for removal. This vertex is removed by calling VERDEL. DELPTN cannot delete nodes that have the status JV(1,IV) = 3. It is therefore recommended that nodes at sharp corners or nodes on important boundaries that are curved, be flagged as JV(1,IV) = 3.

RELAXY

Relaxes the cells that are created in the process of deleting a vertex. The relaxation procedure relocates the designated vertex to the mass center of the surrounding vertices.

LAPLAC

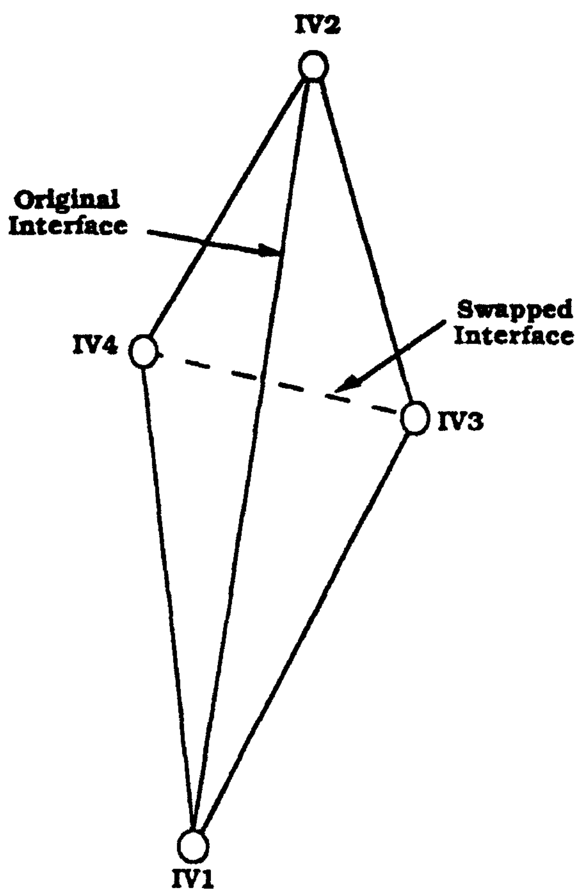
Computes the Laplacian of the pressure and density.

VERDEL

Deletes a designated vertex.

RECNC

Tests the possibility of swapping the designated interface to create two triangles of better quality than the original two.



EOS

Computes γ using to the equation of state for air (Gilmore equation of state), given the density and internal energy of the air. The user may choose to apply the equation of state by setting IOPEOS = 1.

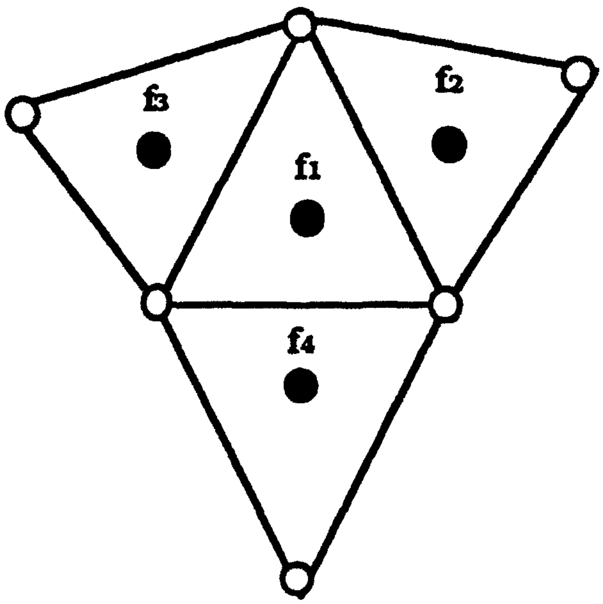
LIFTDR

Computes integral quantity diagnostics on any configuration. The integral quantities are lift, drag, and momentum and are found on boundary interfaces designated as 5.

GRADNT

Computes the gradient of a scalar variable at the center of a cell. It uses a least squares technique to interpolate the values at the center of four triangles (the cell and its three adjacent triangles) to fit (four equations with three unknowns).

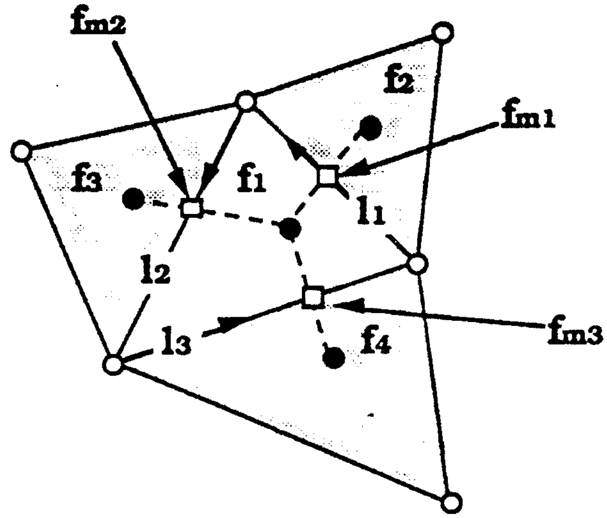
$$f = a_0 + a_1x + a_2y$$



Those gradients are subjected to a monotonicity limiter that ensures no new minima or maxima are produced artificially in the projected values at the interfaces.

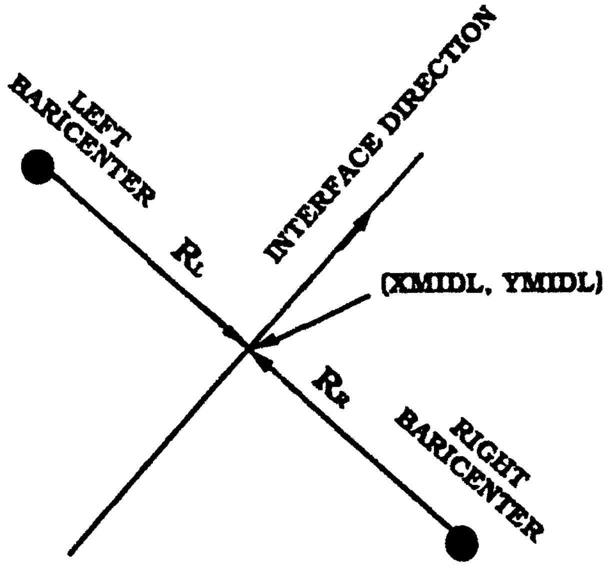
The monotonicity algorithm involves the following steps

1)	find maximum and minimum of f_1, f_2, f_3, f_4 $f_{max} = \mathrm{Max}\,(f_1, f_2, f_3, f_4)$ $f_{min} = \mathrm{Min}\,(f_1, f_2, f_3, f_4)$
2)	compute $\Delta f_{max} = f_{max} - f_1$ $\Delta f_{min} = f_{min} - f_1$
3)	compute incremental projected values at the interfaces $f_{mjR} - f_R = \nabla f_R \cdot \bar{r}_{jR}$ $f_{mjL} - f_L = \nabla f_L \cdot \bar{r}_{jL}$



$$\Delta f_{mjR} = f_{mjR} - f_R = \nabla f_R \cdot \bar{r}_{jR}$$
$$\Delta f_{mjL} = f_{mjL} - f_L = \nabla f_L \cdot \bar{r}_{jL}$$

where j stands for every interface of the cell and fmj is the interpolated value at the middle of the interface.



4) compute the limiter by calculating the minimum of indicator for each edge of the three edges of the cell.

$$\text{right to the interface RUVPR} = \frac{(1 + \text{sign}\ \Delta f_{mjR})\ \Delta f_{max} + (1 - \text{sign}\ \Delta f_{mjR})\Delta f_{min}}{2\ \Delta f_{mjR}}$$

$$\text{left to the interface RUVPL} = \frac{(1 + \text{sign}\ \Delta f_{mjL})\ \Delta f_{max} + (1 - \text{sign}\ \Delta f_{mjL})\Delta f_{min}}{2\ \Delta f_{mjL}}.$$

This formulation ensures that

$$\text{if} \begin{cases} \Delta f_{mj} > 0\ \text{RUVP} = \dfrac{\Delta f_{max}}{\Delta f_{mj}} \\ \Delta f_{mj} < 0\ \text{RUVP} = \dfrac{\Delta f_{min}}{\Delta f_{mj}} \end{cases}$$

the outcome of RUVP is always positive. If RUVP > 1 then the projected value at the interfaces will introduce a new minimum or maximum relative to the values at the baricenters of the appropriate cells.

Select the minimum between the six values for RUVP (two for every one of the three interfaces of the cell) not exceeding unity. The selected minimum

of RUVP is the required limiter. The gradient is multiplied by this limiter that is always less or equal to unity.

FCHART

Computes the projected values at the half time step level based on the local characteristics of the flow. This process extends the accuracy of the code to be second-order in time as well as in space.

The characteristic projection consists of several steps.

1) Calculate the velocity of sound in the two cells bordering the designated interface

CNLEFT = $\sqrt{\gamma_L \cdot P_L/\rho_L}$ sound speed in left cell

CNRIGT = $\sqrt{\gamma_R \cdot P_R/\rho_R}$ sound speed in right cell

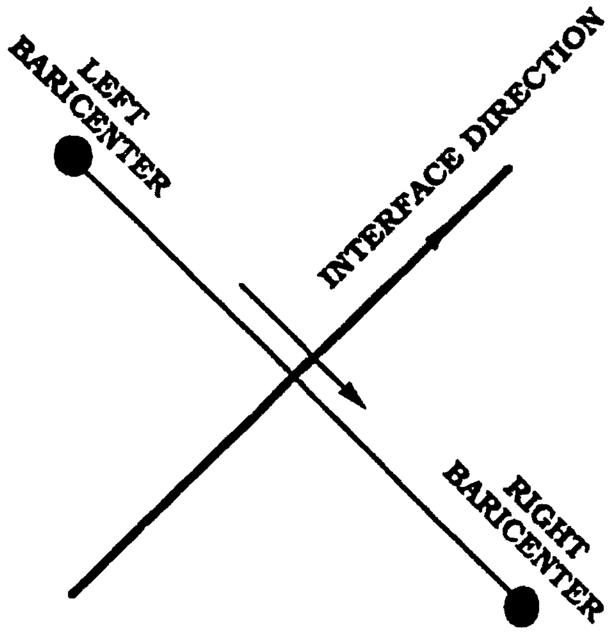
UVLEFT = $\bar{U}_L \cdot \bar{t}$ velocity of fluid at the left cell projected in $\bar{t}$ direction

UVRIGT = $\bar{U}_R \cdot \bar{t}$ velocity of fluid at the right cell projected in $\bar{t}$ direction

where

$$\bar{t} = XXN \cdot \bar{i} + YYN \cdot \bar{j}$$

$$\bar{U} = U \cdot \bar{i} + V \cdot \bar{j}$$



2) To compute the interpolated left and right projected values at time $t^N + \Delta t/2$, we calculate the distances that the disturbances generated from the baricenter of the cells, traveling toward the interface:

$$ZZLEFT = (UVLEFT + CNLEFT) \cdot \Delta t/2$$

$$ZZRIGT = -(UVRIGT - CNRIGT) \cdot \Delta t/2$$

If ZZLEFT or ZZRIGT is negative they are reset to zero.

3) Calculate the distances that the flow will travel if it were to flow at the velocity of each of the local characteristics:

$$ZOLEFT = UVLEFT \cdot \Delta t/2$$

$$ZORIGT = -UVRIGT \cdot \Delta t/2$$

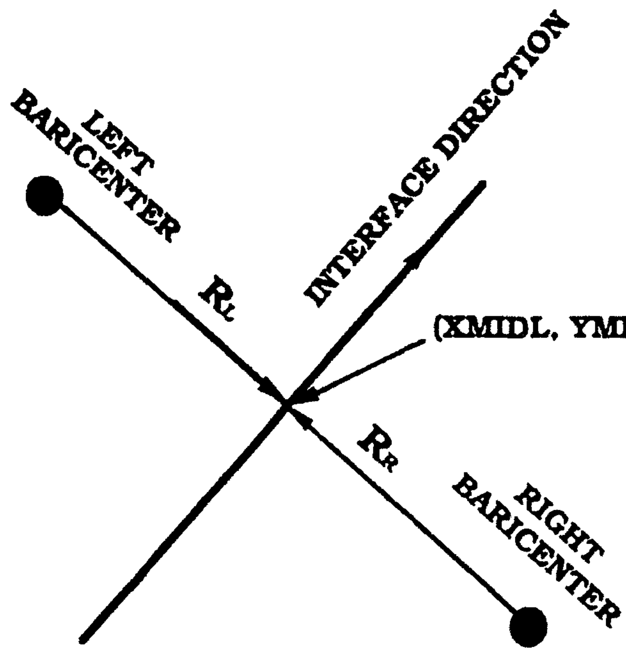
$$ZPLEFT = (UVLEFT + CNLEFT) \cdot \Delta t/2$$

$$ZPRIGT = -(UVRIGT + CNRIGT) \cdot \Delta t/2$$

$$ZMLEFT = (UVLEFT - CNLEFT) \cdot \Delta t/2$$

$$ZMRIGT = -(UVRIGT - CNRIGT) \cdot \Delta t/2.$$

4) Calculate the projected values of the nonconservative variables (density, velocity component (perpendicular and tangential to the interface), and pressure).



For the left cell:

Density	HRRL	=	$\rho_L + \bar{\nabla}\rho_L \cdot (\bar{r}_L - ZZLEFT \cdot \bar{t})$
Perpendicular Velocity	HUUL	=	$U_L + \bar{\nabla}U_L \cdot (\bar{r}_L - ZZLEFT \cdot \bar{t})$

Tangential Velocity	HVVL	=	$V_L + \bar{\nabla} V_L \cdot (\bar{r}_L - \text{ZZLEFT} \cdot \bar{t})$
Pressure	HPPL	=	$P_L + \bar{\nabla} P_L \cdot (\bar{r}_L - \text{ZZLEFT} \cdot \bar{t})$
	GMTLFT	=	$\rho_L \cdot \text{HRRL} \cdot \text{HPPL}$

For the right cell:

Density	HRRR	=	$\rho_R + \bar{\nabla} \rho_R \cdot (\bar{r}_R - \text{ZZRIGT} \cdot \bar{t})$
Perpendicular velocity	HUUR	=	$U_R + \bar{\nabla} U_R \cdot (\bar{r}_R - \text{ZZRIGT} \cdot \bar{t})$
Tangential velocity	HVVR	=	$V_R + \bar{\nabla} V_R \cdot (\bar{r}_R - \text{ZZRIGT} \cdot \bar{t})$
Pressure	HPPR	=	$P_R + \bar{\nabla} P_R \cdot (\bar{r}_R - \text{ZZRIGT} \cdot \bar{t})$
	GMTRGT	=	$\rho_R \cdot \text{HRRR} \cdot \text{HPPR}$

For the left cell, taking into account the following characteristics:

- For UVLEFT + CNLEFT:

$$\text{UUU} = \bar{\nabla} U_L \cdot (\text{ZPLEFT} - \text{ZZLEFT})\, \bar{t}$$

$$\text{PPP} = \bar{\nabla} P_L \cdot (\text{ZPLEFT} - \text{ZZLEFT})\, \bar{t}$$

$$\text{UPLFT} = -0.5 \cdot \left(\text{UUU} + \text{PPP} / \sqrt{\text{GMTLFT}}\right) / \sqrt{\text{GMTLFT}}$$

If UVLEFT + CNLEFT is negative, UPLFT is reset to zero.

- For UVLEFT – CNLEFT:

$$\text{UUU} = \bar{\nabla} U_L \cdot (\text{ZMLEFT} - \text{ZZLEFT}) \cdot \bar{t}$$

$$\text{PPP} = \bar{\nabla} P_L \cdot (\text{ZMLEFT} - \text{ZZLEFT}) \cdot \bar{t}$$

$$\text{UMLFT} = 0.5 \cdot (\text{UUU} - \text{PPP} / \sqrt{\text{GMTLFT}}\ ) / \sqrt{\text{GMTLFT}}$$

If UVLEFT - CNLEFT is negative, UPLFT is reset to zero.

- For UVLEFT:

$$PPP = \bar{\nabla} P_L \cdot (ZOLEFT - ZZLEFT) \cdot \bar{t}$$

$$RRRR = \rho_L + \bar{\nabla}\rho_L \cdot (\bar{r}_L - ZOLEFT) \cdot \bar{t}$$

$$URLFT = PPP/GMTLFT + 1/HRRL - 1/RRRR$$

If UVLEFT is negative, URLEFT is reset to zero.

For the right cell, taking into account the following characteristics:

- For UVRIGT + CNRIGT:

$$UUU = \bar{\nabla} U_R \cdot (ZZRIGT - ZPRIGT)\ \bar{t}$$

$$PPP = \bar{\nabla} P_R \cdot (ZZRIGT - ZPRIGT)\ \bar{t}$$

$$UPRGT = -0.5 \cdot \left(UUU + PPP/\ \sqrt{GMTRGT}\right)/\sqrt{GMTRGT}$$

If UVRIGT + CNRIGT is positive, UMRGT is reset to zero.

- For UVRIGT – CNRIGT:

$$UUU = \bar{\nabla} U_R \cdot (ZZRIGT - ZMRIGT) \cdot \bar{t}$$

$$PPP = \bar{\nabla} P_R \cdot (ZZRIGT - ZMRIGT) \cdot \bar{t}$$

$$UMRGT = 0.5 \cdot \left(UUU - PPP/\sqrt{GMTRGT}\right)/\sqrt{GMTRGT}$$

If UVRIGT - CNRIGT is positive, UMRGT is reset to zero.

- For UVRIGT:

$$PPP = \bar{\nabla} P_R \cdot (ZZRIGT - ZORIGT) \cdot \bar{t}$$

$$RRRR = \rho_R + \bar{\nabla}\rho_R \cdot (\bar{r}_R + ZORIGT) \cdot \bar{t}$$

$$URRGT = PPP/GMTRGT + 1/HRRR - 1/RRRR$$

If UVRIGT - CNRIGT is positive, URRGT is reset to zero.

The projected values will be:

$$\begin{aligned}
RRL &= 1/(1/HRRL - (UPLFT + UMLFT + URLFT)) \\
UUL &= HUUL + (UPLFT - UMLFT)\sqrt{GMTLFT} \\
VVL &= HVVL + (UPLFT - UMLFT)\sqrt{GMTLFT} \\
PPL &= HPPL + (UPLFT + UMLFT)\, GMTLFT \\
RRR &= 1/(1/HRRR - (UPRGT + UMRGT + URRGT)) \\
UUR &= HUUR + (UPRGT - UMRGT)\sqrt{GMTRGT} \\
VVR &= HVVR + (UPRGT - UMRGT)\sqrt{GMTRGT} \\
PPR &= HPPR + (UPRGT + UMRGT) \cdot GMTRGT.
\end{aligned}$$

Those values are the assigned condition for the two sides of the interface. If the interface is a boundary, the right condition is determined according to the type of boundary.

DYNPTN & DYYPTN

DYNPTN applies three distinct criteria to test cells to determine their need for refinement. They are as follows:

For unsteady dynamic simulation

1) total energy flux entering or leaving a cell
2) total density flux entering or leaving a cell
3) density gradient in each cell.

For steady state simulation

1) Pressure gradient in each cell
2) Mach number gradient in each cell
3) density gradient in each cell.

Cells that meet one of those three criteria are flagged, and are actually subdivided in DYYPTN until they meet the area criteria set for refinement (AREADD). The code will compute the maximum of each of the three criteria and set a 5% of the maximum or higher to the refinement criteria for the fluxes and 3% for the gradient. These criteria work extremely well for moving waves.

It should be noted that those error indicators and their levels are set according to the actual simulated condition. For different cases, other error indicators and level settings might be more appropriate than the above.

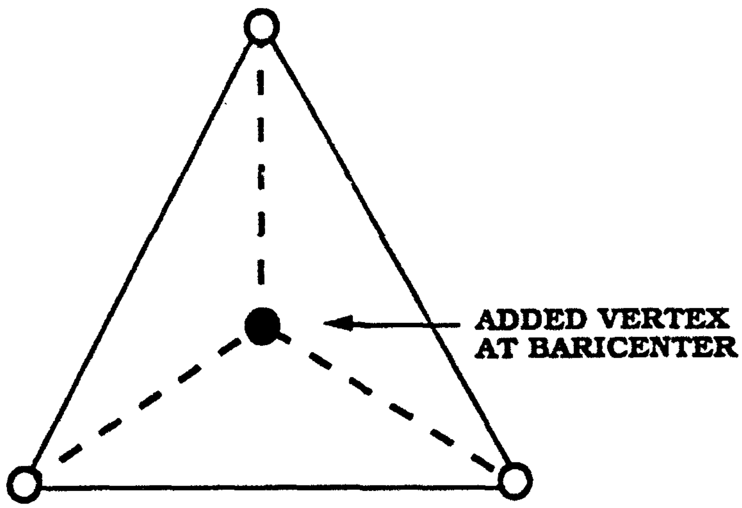
DELPTN

Tests the cells for coarsening criteria. The same criteria that refines the grid are applied to coarsen the grid but in a different setting. Each cell that has less than 5% of the fluxes and less than 3% of the gradient criteria is eligible for coarsening. The code will test the cell flagged for coarsening and will choose one of the three vertices of the cell for deletion by determining which of the three has the smallest aspect ratio. (The aspect ratio is defined as the ratio between the height emerging from the node and its corresponding base.) There are vertices that cannot be removed, such as corners or vertices that preserve the original shape of the boundaries (JV(1,IV) = 3).

After the vertex is deleted, a relaxing procedure is performed on the vertices surrounding the deleted vertex, as well as a swapping procedure to improve the quality of the triangles constructed in the deletion procedure.

VERCEN

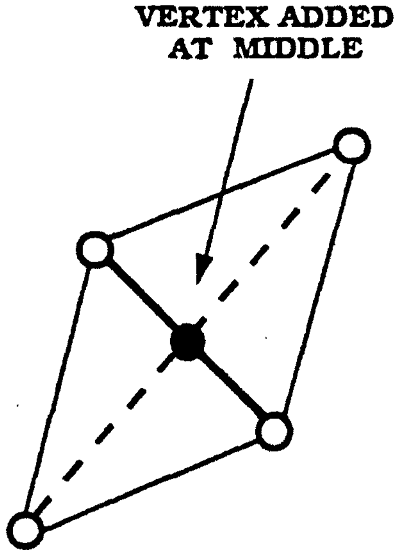
Adds an additional vertex at the baricenter of the designated cell.



VERCEN assigns one of the three new triangles the number of the original triangle and will add two more at the end of cells table. A new vertex plus three new interfaces are added at the end of the associated tables.

DISECT

Adds a new vertex at the middle of the designated edge.



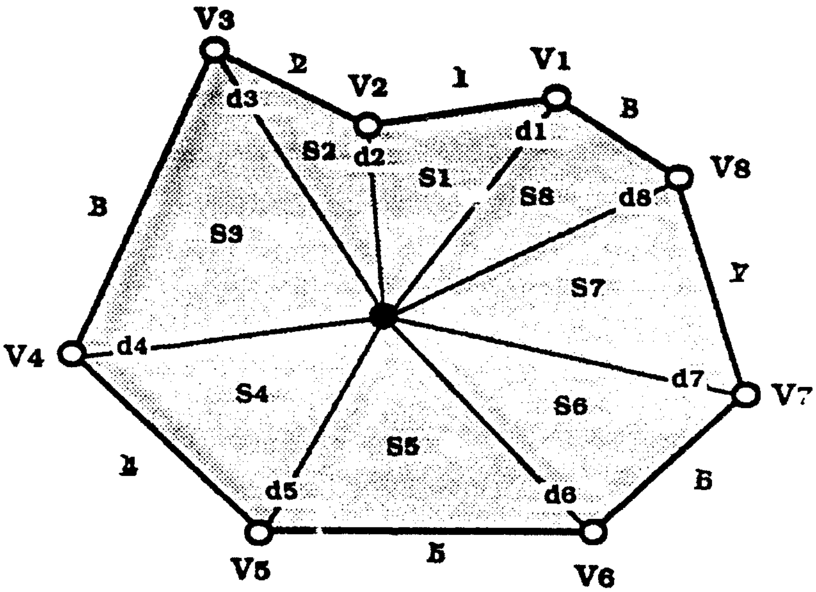
DISECT will add one new vertex, three new edges and two new triangles, all of which are added at the end of the corresponding tables (vertices, edges and cells).

VERDEL

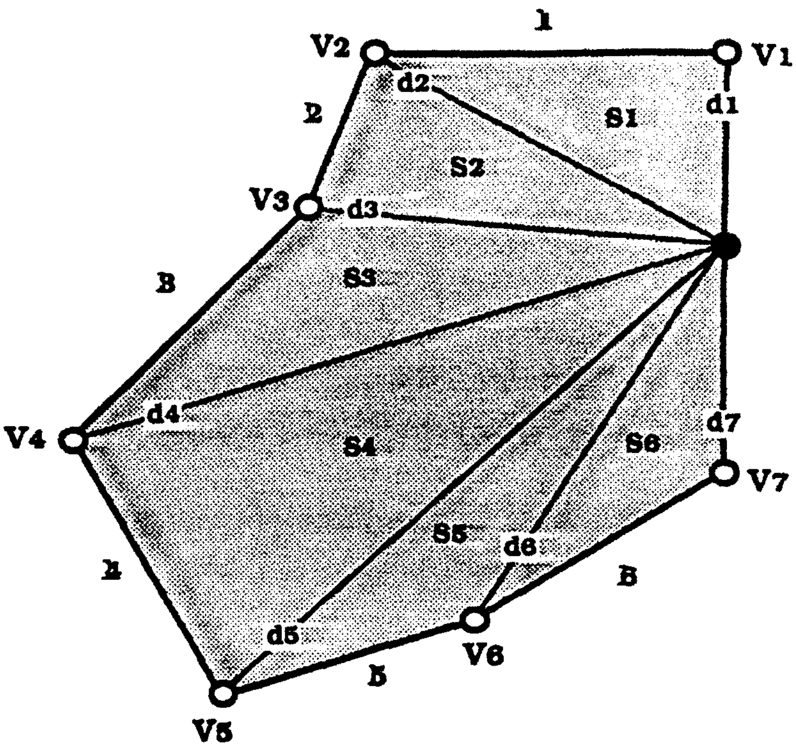
Forces deletion of a designated vertex. There are two types of vertices: deletion of a vertex in the interior of the computational domain and deletion of a vertex on the boundary. The steps of deleting a vertex are:

1) Identify the edges and cells surrounding the designated vertex in the computational domain

Interior Vertex to be Deleted



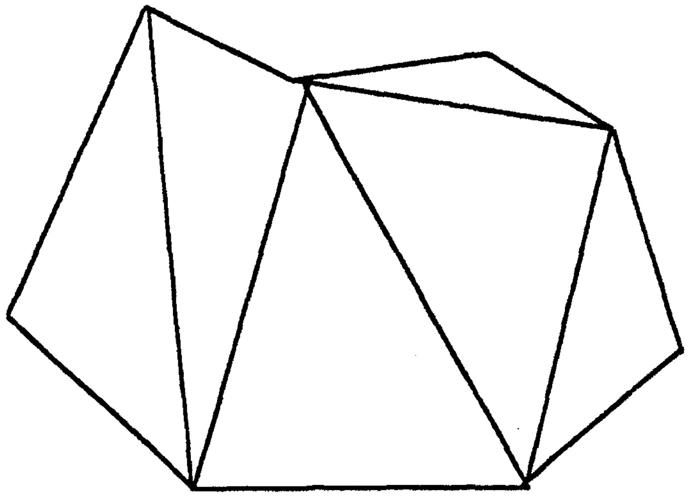
and on the boundary.



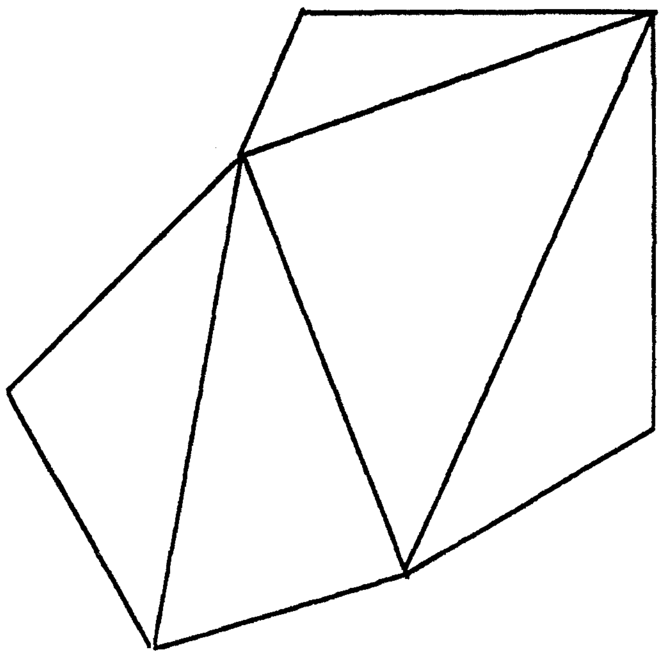
Deletion is more difficult and needs more computational resources than addition. The new vertices edges and cells being added are stacked at the bottom of the corresponding tables while undergoing deletion is always a member in the table. In order not to leave gaps in the table, a more complicated procedure was developed to replace the deleted member by the member at the bottom of the table.

2) Once the vertex, edges and cells joining the designated vertex are deleted we rezone the void (polygon) without adding new vertices. The adding of the new edges and cells are stacking at the end of the corresponding tables.

Interior Vertex



Boundary Vertex



3) A relaxation procedure is performed on the vertices of the polygon (void). This procedure improve the quality of the cells that fill the void.

4) A swap procedure is performed on the new edges that were added in the process of filling the void.

A.1.1 Pre-Processor for the Unstructured Grid

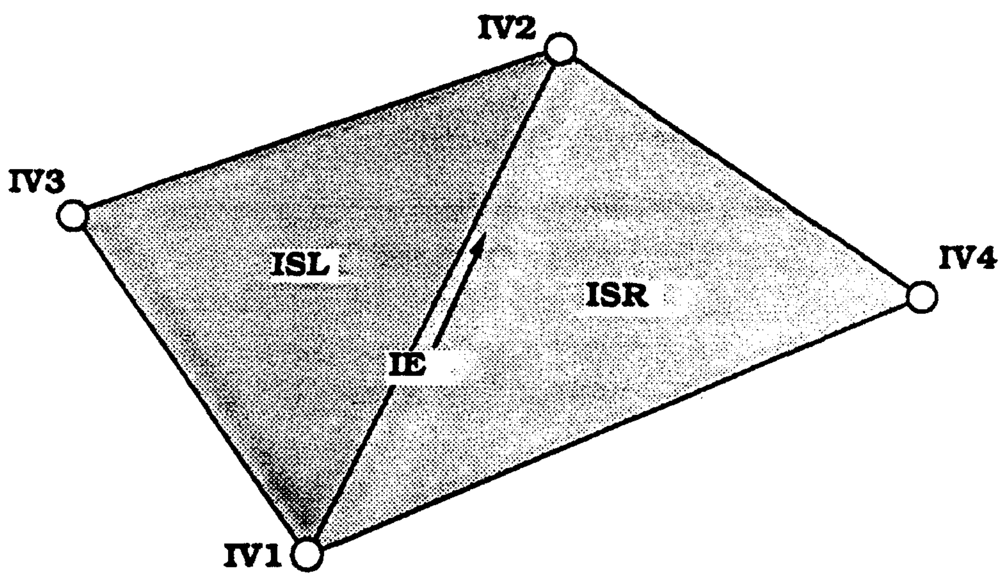
The input geometrical data for AUGUST should provide the ,llowing data:

1) Number of:
   vertices (NV)
   flagged vertices (NVM)
   edges (NE)
   cells (NS)
2) A table of vertices specifying:
   number of vertex (IV)
   x coordinate (XV(1,IV))
   y coordinate (XV(2,IV)).
3) A table of flagged vertices that cannot be removed by the coarsening process:
   number of vertex (IV)
   status of vertex (JV(1, IV))

   The only status of vertex that is currently implemented is the flagging node that does not allow removal:
   JV(1,IV)=3
4) A table of edges specifying:
   number of edges (IE)
   vertex number indicating the beginning of the edge (JE(1,IE))
   vertex number indicating the end of the ·dge (JE(2,IE))
   cell number indicating the cell at the lert the edge (JE(3,IE))
   cell number indicating the cell at the righ the edge (JE(4,IE))
   number associated with the status of the ..ge (JE(5,IE))

If JE(5,IE)=0, the edge is an ordinary edge inside the computational domain.

If JE(5,IE)≠0, the edge lies on the boundary of the domain. The labeling number will indicate what type of boundary to be applied through this edge.



IV1 = JE(1,IE) vertex indicating the beginning of the edge

IV2 = JE(2,IE) vertex indicating the end of the edge

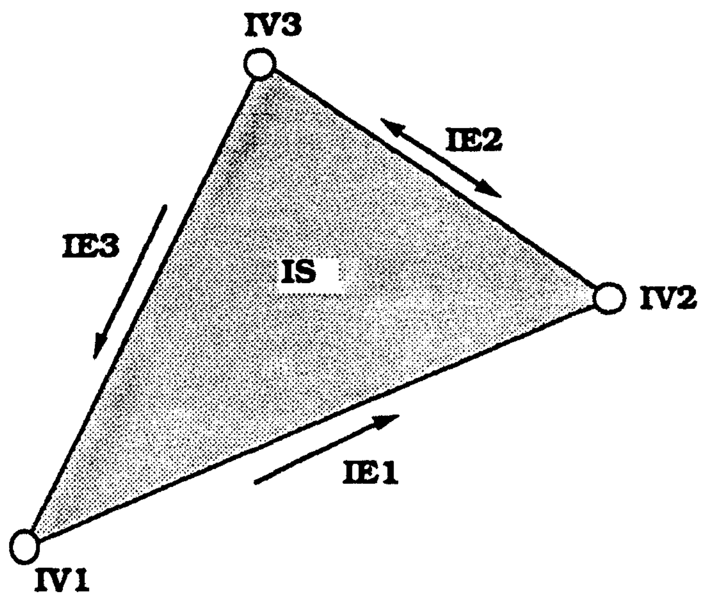
The direction of the edge is defined from IV1 to IV2.

ISL = JE(3,IE)	left triangle
ISR = JE(4,IE)	right triangle
IJE5 = JE(5,IE)	status of the edge
IJE5 = 5	simulating wall conditions
IJE5 = 6	simulating wall conditions
IJE5 = 7	simulating supersonic outlet conditions
IJE5 = 8	simulating supersonic inlet conditions

5) A table of cells specifying:

number of cells (IS)
number of first edge (JS(4,IS))
number of second edge (JS(5,IS))
number of third edge (JS(6,IS))

The sign of JS(4,IS), JS(5,IS), JS(6,IS) indicates whether the direction of the edge is counterclockwise (positive) or clockwise (negative).
The three associated vertices for the triangle JS(1,IS), JS(2,IS), JS(3,IS) are defined by the code in GEOMTR.



The three vertices of the cell are ordered in a counterclockwise arrangement.

IV1 = JS(1,IS)	first vertex
IV2 = JS(2,IS)	second vertex
IV3 = JS(3,IS)	third vertex
IE1 = JS(4,IS)	First edge of the triange directed from IV1 to IV2 (IE1 is positive).
IE2 = JS(5,IS)	Second edge directed originally from IV3 to IV2. (IE2 will be negative because its direction is clockwise)
IE3 = JS(6,IS)	Third edge directed originally from IV3 to IV1 (IE3 is positive)

A.1.2 Post-Processor for the Unstructured Grid

Postprocessing for visualization of the results on an unstructured grid is done in two different codes. The first code, DRAWBF, reads the data as dumped by AUGUST and performs the whole load of computation necessary to produce the information needed for the graphic.

The second code DRAWAF reads the data file written by DRAWBF and uses the DISSPLA software to produce the image on the screen. Breaking the postprocessing job into two separate codes enables the user to run the two codes on different machines.

DRAWBF

Reads an input data file produced by AUGUST and will read another input data file (drawbf.d) specifying the option that the user chooses to have processed.

The input data file drawbf.d specifies the window of the computational domain chosen by the user to be processed. This window is specified by XMIN, XMAX, DX and YMIN, YMAX, DY, where XMIN, XMAX, YMIN, YMAX, will specify the lower and upper limit of the region to be drawn. DX and DY will be parameters for DISSPLA to subdivide the axis into tick marks.

DISSPLA is constrained to seven colors. To extend the number of contour levels, the code can be set to draw a couple of levels in each color (7 x **NLEV** where **NLEV** is the number of levels for each color).

The user should specify the variable he wants to draw:

IHYD = 1 is density,
= 2 is velocity in the x direction
= 3 is velocity in the y direction
= 4 is pressure
= 5 is gamma (γ)
= 6 is Mach number
= 7 is entropy
= 8 is a vector plot of the velocity field
= 9 is a plot of the location of particles

The last parameter that the user should specify is IREC. IREC specifies how many dumps are in the input file produced by AUGUST. If IREC=0, the user will get as many figures as the number of dumps produced by AUGUST. Otherwise, the user will get the figure corresponding to IREC specified in the input file.

Subroutine NEXTREC reads a whole dump from the input file (written by AUGUST). It will make sure that the allocation of memory is adequate according to the number of vertices, edges and triangles to be processed. If the memory allocation is not adequate, the code will stop with an explanatory message.

Subroutine LOADF loads the portion of data needed according to the specification of the window and according to the specified IHYD into the appropriate matrices in the code.

Subroutine PHYDR produces the data for the contour plots.

Subroutine VECTOR produces the data for the vector plot of the velocity field.

Subroutine TRACER produces the data for the location of particles.

DRAWAF

DRAWAF reads an input data file (drawbf.k) produced by DRAWBF and another input file (drawaf.d) that specifies the format chosen for display.

The parameters specified in drawaf.d are:

IFMESH = 0: No grid is drawn.

IFMESH = 1: Grid is drawn.

IOPTON = 0: A single frame is drawn.

IOPTON = 1: Two frames are drawn, one for the grid and one for displaying results. The frame for the grid is drawn even if IFMESH=0, but in this case the frame will stay empty.

IOPTON = 2: Identical with IOPTN=1 except the level on the bar chart is written in engineering format (XE+Y). As in the former, it is written keeping a four decimal digit.

ICONFG = 0: The basic dimension for the frames is specified as 6.0 x 3.0 inches (in the x and y axis, respectively). The code makes sure that the proportionality of the frame matches the physical window to be drawn, so that the figure will not be distorted. This is done by redefining the x or y dimension of the frame accordingly, but not to exceed the 6.0 x 3.0 on the screen (ICONFG=0 should be picked if IOPTON > 0 and a two-frame drawing is desired).

ICONFG>0: The same as ICONFG=0 except that the basic dimensions are defined now as 6.0 x 6.0 inches. This option should be specified if a one frame drawing is desired.

Caption 1
Caption 2

The user can specify a header for the drawing composed of two lines to be specified as Caption 1 and Caption 2 in the input file.

The standard drawing includes the number of vertices, edges and cells as well as the Mach number, lift, drag, moment, angle of attack (for drawing diagnostics for a wing profile). An indication of the nature of the results that appear on the drawing is also included, i.e., the physical variables drawn are identified by the parameter passing from DRAWBF.

It should be noted that the format of the output drawing is very easily redesigned to meet the needs of an individual user.

1. Read geometrical data defining the initial grid. The current format is set to read data file from Smart (two dimension grid generator).
2. Read geometrical data defining the grid read from a file dumped by a previous run of the code.
3. Initialize the physical variables according to IOPTN (either steady state or moving shock wave). If a different initial setting is needed, it should replace the current setting.
4. Read the physical variables from a file dumped by a previous run.

A.2 AUGUSTT (3D)

The subroutines in the AUGUSTT code are organized here as they appear in the listing in Appendix B. A brief description indicates the function performed by each subroutine.

TABLE A.2.1

LIST OF SUBROUTINES

The subroutines in the AUGUST code are organized here as they appear in the listing in Appendix B. A brief description indicates the function performed by each subroutine.

1. MAIN	Governing program for AUGUST. Reads input files and sets the mode for the computation.
2. HYDRFL	Computes the fluxes at interfaces by applying the Godunov algorithm to solve the Riemann problem across the interface.
3. HYDRMN	Controls the computation. The integration of the fluxes and update of the physical variables and writing to output files are performed in this subroutine.
4. GEOMTR	Calculates the geometrical quantities not provided by the input data file but needed for the computational algorithm. GEOMTR is only used once for starting a new simulation.
5. UPDATE	Reads the input file for a new simulation and calls GEOMTR to update the geometrical variables needed to perform the computation.
6. UPGRAD	Called if a restart run is performed. Will read the appropriate file written at the end of the previous run.

7. GRADNT	Computes the gradient of the physical variables to improve the prediction of those variables for the two sides of the interface. The gradients are subjected to the monotonicity condition that limits the projected values, thus preventing new maxima-minima to be caused artificially by interpolation (IOPORD = 2). Calls FCHART in order to compute projected values at the half timestep associated with the local characteristics of the flow.
8. FIRST	The equivalent of GRADNT if run in a first order mode (IOPORD = 1). Using FIRST assumes that the physical variables are constant in each cell. Takes care of the boundary conditions if the interface is a boundary.
9. FCHART	Computes the projected values at a half timestep for the two sides of the interface based on the local characteristics of the flow. Called by GRADNT, it modifies the projected values for the two sides of the interface and assigns them to the correct location in memory. Takes care of the boundary conditions if the interface is a boundary.

The MAIN Program

All of the data input and initiation of a run (or a restart run) is performed in MAIN. The actual simulation is controlled by HYDRMN, which is called from MAIN. At the completion of a run, control is returned to MAIN and a successful termination prints the message STOP 777.

MAIN contains one name list (file no. 2) and requires an input file that contains the grid data description (file no. 16). The data organization for the

grid file is described in Appendix A. The following files should be included: DMSH00.H, DPHSϕ0.H, DHYD00.H.

NAMELIST/DATA	ICOND	ICONP	IOPTN
	XMCHIN	RIN	PIN
	ALFA	HRGG	IHRN
	NTIME	NDUMP	IOPORD

VARIABLE	PURPOSE
ICOND	= 0 READ INPUT GRID FOR A NEW SIMULATION = 1 READ THE GRID FROM PREVIOUS RUN

ICOND = 0: MAIN will read the initial grid definition stored in file number 16. The current setting is to read the input file as provided by Smart, a two-dimensional triangular grid generator that runs interactively on a Macintosh personal computer.

MAIN will call UPDATE, which will call GEOMTR. GEOMTR will compute essential geometrical parameters that are not provided by file 16. All geometrical information is dumped into output files (8 and 88) so that ICOND=0 is used only once at the beginning of a new simulation.

ICOND = 1: MAIN will call UPGRAD, which will call one of the output files (8 or 88) written by the previous run. This will load the geometrical definition of the grid (either 8 or 88---they are identical). Writing identical files provides a backup in the event that the job terminates for lack of time while in the process of writing to one of those output files.

VARIABLE	PURPOSE
ICONP	= 0 PRIMITIVE VARIABLES INITIALIZED = 1 VARIABLES READ FROM PREVIOUS RUN

ICONP = 0: Initialize the primitive variables in computational domain with an initial value specified by the user. The two options set by the code are controlled by IOPTN.

ICONP = 1: The flow field condition reads in files 8 or 88 and provides a follow-up run set from the previous run.

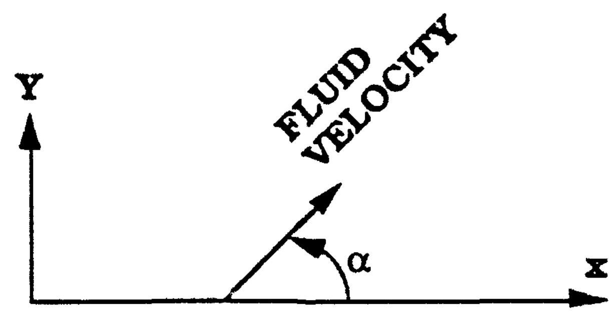
VARIABLE	PURPOSE
IOPTN	= 1 SOLUTION FOR STEADY STATE = 2 SOLUTION FOR TRANSIENT PHENOMENA

There are two choices available to set the initial condition of the problem.

IOPTN = 1: Assign the conditions at the inlet to the computational domain. This is the fastest way to get a steady-state solution for the conditions specified at the inlet. In this option, PIN (pressure), RIN (density) and XMCHIN (Mach number) are assigned to the pressure density and velocity (the speed of sound is computed in the code) and imposed at the inlet boundaries.

IOPTN = 2: Used if a shock wave is to be simulated moving from the inlet (edge boundary 8) to the outlet (edge boundary 7). For this setting, specify PIN (ambient pressure in the chamber), RIN (ambient density in the chamber) and XMCHIN (upstream Mach number). The code will use the normal shockwave relations for an adiabatic flow of a completely perfect fluid to compute the static-pressure ratio across the shock P2/P1 and the density ratio r2/r1, and the ratio of the Mach number across the shock M2/M1. These computed quantities are applied to set correctly the condition on the pressure density and velocity at the inlet boundary.

VARIABLE	PURPOSE
ALFA	THE DIRECTION OF INFLOW IN DEGREES RELATIVE TO A RIGHT-HAND COORDINATE SYSTEM. ALFA = 0 MEANS FLOW FROM LEFT TO RIGHT.



The velocity computed by the code according to the input data provided by the user is split (projected) in the X and Y directions by using α.

VARIABLE	PURPOSE
HRGG	INITIAL γ IN THE EQUATION OF STATE. THE CODE RUNS USING THE IDEAL EQUATION OF STATE AS A BASELINE AND SHOULD BE MODIFIED IF SOMETHING ELSE IS DESIRED. IOPEOS=1 WILL TRIGGER THE USE OF GILMORE EQUATION OF STATE.

VARIABLE	PURPOSE
IHRN	NUMBER OF ITERATIONS IN THE RIEMANN SOLVER TO FIND THE DIAPHRAGM SOLUTION. (THREE TO FOUR SHOULD BE USED AND THE NUMBER INCREASED ONLY FOR VERY HIGH MACH NUMBER CASES.)

VARIABLE	PURPOSE
NTIME	NUMBER OF REPEATS FOR THE INTEGRATION SEQUENCE. AN OUTPUT DUMP IS DONE FOR EVERY SEQUENCE REPEAT.

VARIABLE	PURPOSE
NDUMP	NUMBER OF OUTER LOOP ITERATIONS IN THE CALCULATION WHERE REFINING IS DONE FOR EVERY SEQUENCE REPEAT WITHOUT COARSENING.

VARIABLE	PURPOSE
IOPORD	= 1 THE CODE WILL RUN FIRST ORDER GODUNOV METHOD = 2 THE CODE WILL RUN SECOND ORDER GODUNOV METHOD

IOPORD = 1 Subroutine FIRST is called.

IOPORD = 2 Subroutine GRADNT is called.

HYDRFL

Computes the fluxes across interfaces when the conditions for both sides are given. The fluxes are computed assuming a shock solution at a ruptured diaphragm simulated by the presence of the interface. The conditions existing on the two sides of the diaphragm will define the condition of the flow at the diaphragm location. These conditions are computed by solving the Riemann problem using the Godunov algorithm. The condition at the diaphragm defines the flux of energy, mass, and momentum passing across the interface. The Euler conservation law is applied to conserve energy, mass, and momentum crossing interfaces from one cell to the other.

Quantity	Side 1	Diaphragm (Interface)	Side 2
Density	ρ_1	ρ	r_2
Pressure	P_1	P	P_2
Velocity Perpendicular to Interface	u_1	u	u_2
Velocity Parallel to Interface	v_1	v	v_2
Velocity Parallel to Interface to Construct a Right-Hand Coordinate System (u, v, w,)	w_1	w	w_2

HYDRMN

Controls the code and the iteration loops. It calls HYDRFL to find the interface fluxes. These fluxes are integrated to update the physical variables in each cell. If adaptation of the grid is required, HYDRMN also controls the output by writing the necessary information on files for postprocessing data and for restarting the AUGUST code at a later time. It also manages print file diagnostics.

GEOMTR

Calculates geometrical variables that are not supplied by the input data and are needed to run the code. For example, it computes:

1) distances between baricenters of adjoining cells;
2) the location of the intersection between the line joining adjacent baricenter cells and the interface.

The code changes the direction of the boundary edges so that all are arranged counter clockwise and the associated computational cell is always on the left side. GEOMTR is called once in the beginning of a new simulation.

UPDATE

Called in the beginning of a new simulation for setting geometrical variables not provided by the input data. (It calls GEOMTR.)

UPGRAD

Called if the run is a restart. UPGRAD will read the appropriate file (either 8 or 88) dumped by the previous run.

GRADNT

Compute the gradients of the physical variables in each cell. These computed gradients, along with the physical values at the baricenters, are applied using linear interpolation to predict the values on the interface.

The computed gradients are subjected to the monotonicity condition, ensuring that the projected values are bounded by the value of each quantity in the three adjacent cells, and to make sure that no new maxima or minima occur. The projection of quantities to the interface improves the results from the code and provides second order accuracy in space.

GRADNT calls FCHART, which computes the projected values at the interfaces at the half timestep level according to the local characteristics of the flow in each cell bordering the interface cell. The assignment of values at the two sides of each interface is done at the end of FCHART. This same loop also imposes the boundary conditions for the interfaces at the boundaries of the computational domain.

FIRST

Assigns flow quantities to each side of an edge. These are based on the values at the baricenter of the triangles on either side of the edge. FIRST uses a first order approximation to find the values at the edge.

The user can specify FIRST or GRADNT by choosing 1 or 2 for the parameter IOPORD.

FCHART

Called by GRADNT to compute the values projected at the interfaces at the half timestep. These calculations are done by applying the local velocity characteristics in each cell. This projection in time improves the results and makes the code second order accurate in time.

GRADNT

Computes the gradient of a scalar variable at the center of a cell. The gradient theorem is applied for each cell.

$$\int_{\text{volume}} \nabla \cdot dv = \oint_{\text{four surfaces}} f \, \bar{n} \, ds$$

Those gradients are subjected to a monotonicity limiter that ensures no new minima or maxima are produced artificially in the projected values at the interfaces.

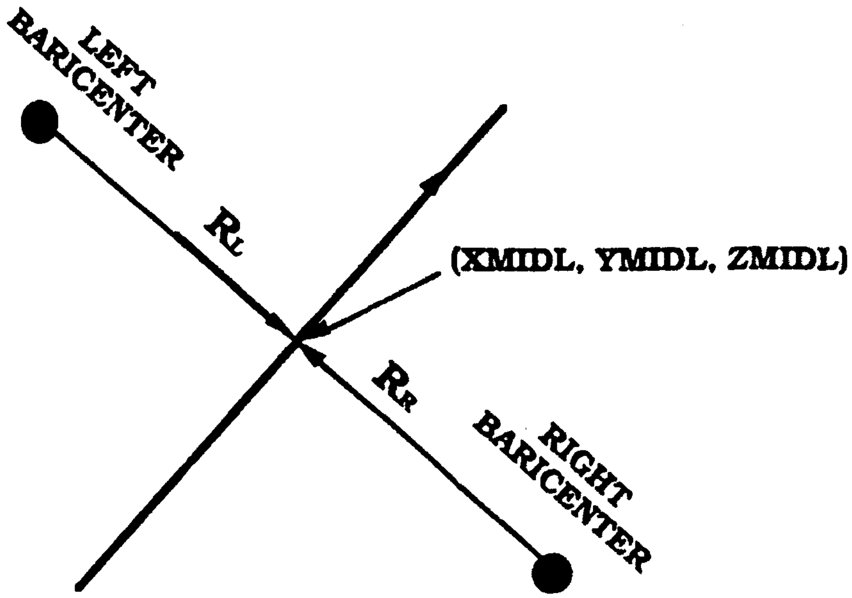
The monotonicity algotithm involves the following steps.

1)	find maximum and minimum of f_1, f_2, f_3, f_4, f_5 $f_{max} = \text{Max}\,(f_1, f_2, f_3, f_4, f_5)$ $f_{min} = \text{Min}\,(f_1, f_2, f_3, f_4, f_5)$
2)	compute $\Delta f_{max} = f_{max} - f_1$ $\Delta f_{min} = f_{min} - f_1$
3)	compute incremental projected values at the interfaces $f_{mjR} - f_R = \bar{\nabla} f_R \cdot \bar{r}_{jR}$ $f_{mjL} - f_L = \bar{\nabla} f_L \cdot \bar{r}_{jL}$

$$Df_{mjR} = f_{mjR} - f_R = \bar{\nabla} f_R \cdot \bar{r}_{jR}$$

$$Df_{mjL} = f_{mjL} - f_L = \bar{\nabla} f_L \cdot \bar{r}_{jL}$$

where j stands for every interface of the cell and fmj is the interpolated value at the middle of the interface.



4) compute the limiter by calculating the minimum of indicator for each edge of the four surfaces of the cell.

$$\text{right to the interface } RUVPR = \frac{(1 + \text{sign } \Delta f_{mjR}) \, \Delta f_{max} + (1 - \text{sign } \Delta f_{mjR}) \Delta f_{min}}{2 \, \Delta f_{mjR}}$$

$$\text{left to the interface } RUVPL = \frac{(1 + \text{sign } \Delta f_{mjL}) \, \Delta f_{max} + (1 - \text{sign } \Delta f_{mjL}) \Delta f_{min}}{2 \, \Delta f_{mjL}}.$$

This formulation ensures that:

$$\text{if} \begin{cases} \Delta f_{mj} > 0 \; RUVP = \dfrac{\Delta f_{max}}{\Delta f_{mj}} \\ \Delta f_{mj} < 0 \; RUVP = \dfrac{\Delta f_{min}}{\Delta f_{mj}} \end{cases}.$$

the outcome of RUVP is always positive. If RUVP > 1 then the projected value at the interfaces will introduce a new minima or maxima as compared to the values at the baricenters of the appropriate cells.

Select the minimum between the six values for RUVP (two for every one of the three interfaces of the cell) not exceeding unity. The selected minimum of RUVP is the required limiter. The gradient is multiplied by this limiter that is always less or equal to unity.

ECHART

Computes the projected values at the half timestep level based on the local characteristics of the flow. This process extends the accuracy of the code to be second-order in time as well as in space.

The characteristics projection consists of several steps.

1) Calculate the velocity of sound in the two cells bordering the designated interface:

CNLEFT = $\sqrt{\gamma_L \cdot P_L/\rho_L}$ sound speed in left cell

CNRIGT = $\sqrt{\gamma_R \cdot P_R/\rho_R}$ sound speed in right cell

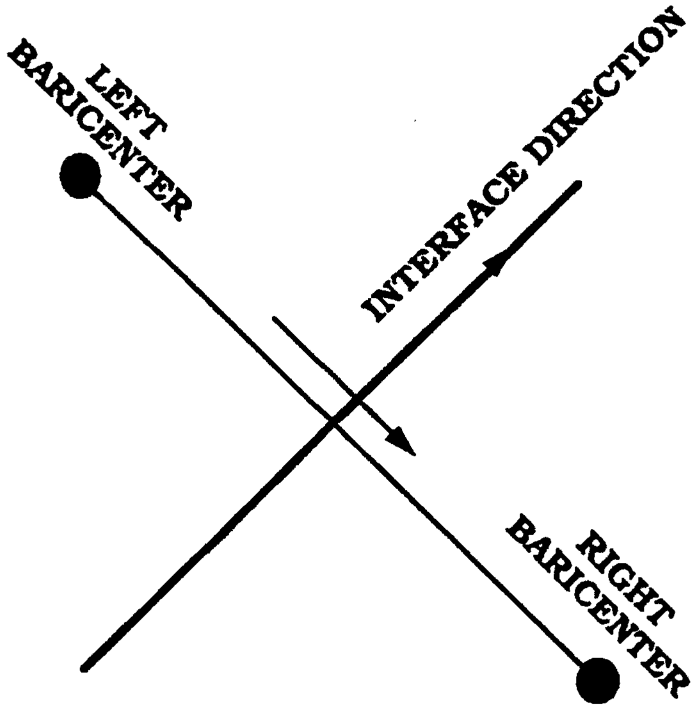
UVLEFT = $\bar{UL} \cdot \bar{t}$ velocity of fluid at the left cell projected in $\bar{t}$ direction

UVRIGT = $\bar{UR} \cdot \bar{t}$ velocity of fluid at the right cell projected in $\bar{t}$ direction

where:

$$\bar{t} = XXN \cdot \bar{i} + YYn \cdot \bar{j} + zzn\, \bar{k}$$

$$\bar{U} = U \cdot \bar{i} + v \cdot \bar{j} + w \cdot \bar{k}$$



2) To compute the interpolated left and right projected values at time tN + Dt/2, we calculate the distances that the disturbances generated from the baricenter of the cells, traveling toward the interface:

$$ZZLEFT = (UVLEFT + CNLEFT) \cdot \Delta t/2$$

$$ZZRIGT = - (UVRIGT - CNRIGT) \cdot \Delta t/2$$

If ZZLEFT or ZZRIGT are negative they are reset to zero.

3) Calculate the distances that the flow will travel if it were to flow at the velocity of each of the local characteristics:

$$ZOLEFT = UVLEFT \cdot \Delta t/2$$

$$ZORIGT = - UVRIGT \cdot \Delta t/2$$

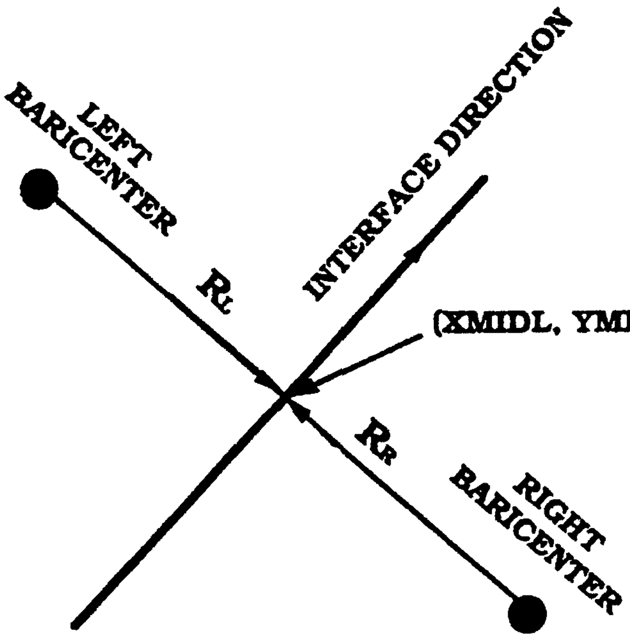
$$ZPLEFT = (UVLEFT + CNLEFT) \cdot \Delta t/2$$

$$ZPRIGT = - (UVRIGT + CNRIGT) \cdot \Delta t/2$$

$$\text{ZMLEFT} = (\text{UVLEFT} - \text{CNLEFT}) \cdot \Delta t/2$$

$$\text{ZMRIGT} = -(\text{UVRIGT} - \text{CNRIGT}) \cdot \Delta t/2.$$

4) Calculate the projected values of the nonconservative variables (density, velocity component (perpendicular and tangential to the interface), and pressure).



For the left cell:

Density	HRRL	=	$\rho_L + \bar{\nabla}\rho_L \cdot (\bar{r}_L - \text{ZZLEFT} \cdot \bar{t})$
Perpendicular Velocity	HUUL	=	$U_L + \bar{\nabla}U_L \cdot (\bar{r}_L - \text{ZZLEFT} \cdot \bar{t})$
Tangential Velocity	HVVL	=	$V_L + \bar{\nabla}V_L \cdot (\bar{r}_L - \text{ZZLEFT} \cdot \bar{t})$
Pressure	HPPL	=	$P_L + \bar{\nabla}P_L \cdot (\bar{r}_L - \text{ZZLEFT} \cdot \bar{t})$
	GMTLFT	=	$\rho_L \cdot \text{HRRL} \cdot \text{HPPL}$

For the right cell:

Density	HRRR	=	$\rho_R + \bar{\nabla}\rho_R \cdot (\bar{r}_R - ZZRIGT \cdot \bar{t})$
Perpendicular velocity	HUUR	=	$U_R + \bar{\nabla}U_R \cdot (\bar{r}_R - ZZRIGT \cdot \bar{t})$
Tangential velocity	HVVR	=	$V_R + \bar{\nabla}V_R \cdot (\bar{r}_R - ZZRIGT \cdot \bar{t})$
Pressure	HPPR	=	$P_R + \bar{\nabla}P_R \cdot (\bar{r}_R - ZZRIGT \cdot \bar{t})$
	GMTRGT	=	$\rho_R \cdot HRRR \cdot HPPR$

For the left cell, taking into account the following characteristics:

- For (UVLEFT + CNLEFT):

  $UUU = \bar{\nabla}U_L \cdot (ZPLEFT - ZZLEFT)\,\bar{t}$

  $PPP = \bar{\nabla}P_L \cdot (ZPLEFT - ZZLEFT)\,\bar{t}$

  $UPLFT = -0.5 \cdot \left(UUU + PPP/\sqrt{GMTLFT}\right) / \sqrt{GMTLFT}$

  If UVLEFT + CNLEFT is negative, UPLFT is reset to zero.

- For UVLEFT − CNLEFT:

  $UUU = \bar{\nabla}U_L \cdot (ZMLEFT - ZZLEFT) \cdot \bar{t}$

  $PPP = \bar{\nabla}P_L \cdot (ZMLEFT - ZZLEFT) \cdot \bar{t}$

  $UMLFT = 0.5 \cdot (UUU - PPP/\sqrt{GMTLFT}\ ) / \sqrt{GMTLFT}$

  If UVLEFT - CNLEFT is negative, UPLFT is reset to zero.

- For UVLEFT:

  $PPP = \bar{\nabla}P_L \cdot (ZOLEFT - ZZLEFT) \cdot \bar{t}$

$$RRRR = \rho_L + \bar{\nabla}\rho_L \cdot (\bar{r}_L - ZOLEFT) \cdot \bar{t}$$

$$URLFT = PPP/GMTLFT + 1/HRRL - 1/RRRR$$

If UVLEFT is negative, URLEFT is reset to zero.

For the right cell, taking into account the following characteristics:

- For UVRIGT + CNRIGT:

$$UUU = \bar{\nabla}U_R \cdot (ZZRIGT - ZPRIGT)\, \bar{t}$$

$$PPP = \bar{\nabla}P_R \cdot (ZZRIGT - ZPRIGT)\, \bar{t}$$

$$UPRGT = -0.5 \cdot \left(UUU + PPP/\sqrt{GMTRGT}\right) / \sqrt{GMTRGT}$$

If UVRIGT + CNRIGT is positive, UMRGT is reset to zero.

- For UVRIGT – CNRIGT:

$$UUU = \bar{\nabla}U_R \cdot (ZZRIGT - ZMRIGT) \cdot \bar{t}$$

$$PPP = \bar{\nabla}P_R \cdot (ZZRIGT - ZMRIGT) \cdot \bar{t}$$

$$UMRGT = 0.5 \cdot \left(UUU - PPP/\sqrt{GMTRGT}\right) / \sqrt{GMTRGT}$$

If UVRIGT - CNRIGT is positive, UMRGT is reset to zero.

- For UVRIGT:

$$PPP = \bar{\nabla}P_R \cdot (ZZRIGT - ZORIGT) \cdot \bar{t}$$

$$RRRR = \rho_R + \bar{\nabla}\rho_R \cdot (\bar{r}_R + ZORIGT) \cdot \bar{t}$$

$$URRGT = PPP/GMTRGT + 1/HRRR - 1/RRRR$$

If UVRIGT - CNRIGT is positive, URRGT is reset to zero.

The projected values will be:

$$RRL = 1/(1/HRRL - (UPLFT + UMLFT + URLFT))$$

$$UUL = HUUL + (UPLFT - UMLFT)\sqrt{GMTLFT}$$

$$VVL = HVVL + (UPLFT - UMLFT)\sqrt{GMTLFT}$$
$$PPL = HPPL + (UPLFT + UMLFT)\ GMTLFT$$
$$RRR = 1/(1/HRRR - (UPRGT + UMRGT + URRGT))$$
$$UUR = HUUR + (UPRGT - UMRGT)\sqrt{GMTRGT}$$
$$VVR = HVVR + (UPRGT - UMRGT)\sqrt{GMTRGT}$$
$$PPR = HPPR + (UPRGT + UMRGT) \cdot GMTRGT.$$

Those values are the assigned condition for the two sides of the interface. If the interface is a boundary, the right condition is determined according to the type of boundary.

A.2.1 Preprocessor for the Three-Dimensional Unstructured Grid

The input geometrical data for AUGUST should provide the following data:

1) Number of vertices (NV)

2) A table of vertices specifying:
   number of vertex (IV)
   x coordinate (XV(1,IV))
   y coordinate (XV(2,IV))
   z coordinate (XV(3,IV).

3) Number of edges (NE)

4) A table of edges specifying
   number of edges (IE)
   vertex number indicating the beginning of the edge (JE(1,IE))
   vertex number indicating the end of the edge (JE(2,IE))

IV1 =	JE(1,IE)	vertex indicating the beginning of the edge
IV2 =	JE(2,IE)	vertex indicating the end of the edge

The direction of the edge is defined from IV1 to IV2.

5) Number of sides (NS)

6) A table of sides (triangles) specifying:

number of sides (IS)
number of first vertice (JS(1,IS))
number of second vertices (JS(2,IS))
number of third vertices (JS(3,IS))
number of first edge (JS(4,IS))
number of second edge (JS(5,IS))
number of third edge (JS(6,IS))

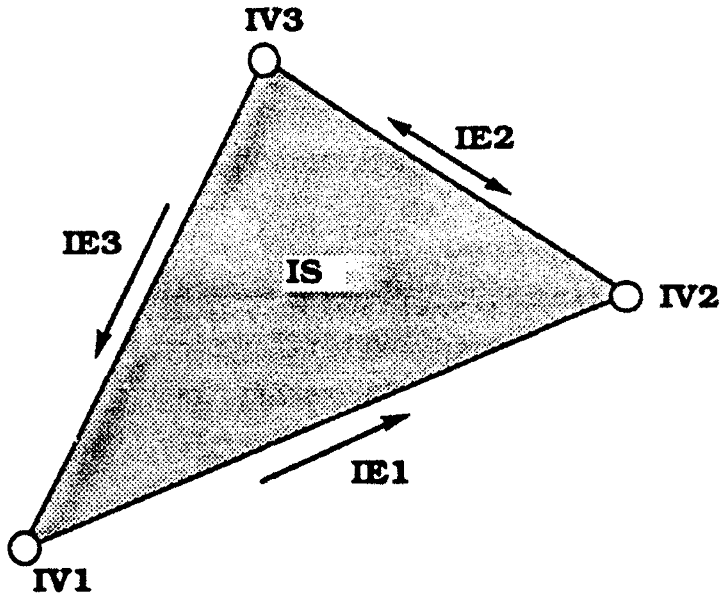
The sign of JS(4,IS), JS(5,IS), JS(6,IS) indicates whether the direction of the edge is counter clockwise (positive) or clockwise (negative).

tetrahedra on left to the side (JS(7,IS))
tetrahedra on right to the side (JS(8,IS))

Number associated with the status of the side (JS(9,IS)).

if JS(9,IS) = 0 the side is an ordinary side inside the computational domain.

if JS(9,IS) ≠ 0 the side lies on the boundary of the domain. The labeling number will indicate what type of boundary to applied through this side.



The three vertices of the side are ordered in a counter clockwise arrangement.

IV1 = JS(1,IS)	first vertex
IV2 = JS(2,IS)	second vertex
IV3 = JS(3,IS)	third vertex
IE1 = JS(4,IS)	First edge of the triangle directed from IV1 to IV2 (IE1 is positive).
IE2 = JS(5,IS)	Second edge directed originally from IV3 to IV2. (IE2 will be negative because its direction is clockwise.)
IE3 = JS(6,IS)	Third edge directed originally from IV3 to IV1 (IE3 is positive).
IC1 =JS(7,IS)	tetrahedra on the left
IC2 = JS(8,IS)	tetrahedra on the right

The normal to the side is directed from IC1 toward IC2. If the side is a boundary, the normal is always from the computational domain pointing outside (out of the fluid domain). The three vertices are ordered in a counter clockwise direction opposite to the direction of the normal to the side. For a boundary side, IC2 will be always zero.

IJS = JS(9,IS) Status of the side

IJS9 = 6 Simulating wall conditions

IJS9 = 7 Simulating supersonic outlet conditions

IJS9 = 8 Simulating supersonic inlet conditions.

7) A table of sides specifying:
x coordinate of baricenter of side (XS(1,IS))
y coordinate of baricenter of side (XS(2,IS))
z coordinate of baricenter of side (XS(3,IS))
area of side (XS(4,IS))

8) A table of sides specifying:
the three component of the vector normal to the side:

$$\bar{N} = XN(IS)\,\vec{i} + yN(IS)\,\bar{J} + ZN(IS)\bar{k}$$

the three component of the parallel vector tangential to the side:

$$\vec{P} = XP(IS)\,\vec{r} + YP(IS) + ZP(IS)\vec{k}$$

the three component of the parallel vector tangential to the side:

$$T = XT(IS)\,\vec{r} + YT(IS)\,\vec{j} + ZT(IS)\,\vec{k}$$

where $\bar{P} \times \bar{T} = \bar{N}$ (the normal, perpendicular and parallel vectors form a local right-handed coordinate system).

9) number of cells (tetrahedrals) (NC)

10) A table of cells specifying:
Number of cells (IC)
Number of first vertex (JC(1,IC))
Number of second vertex (JC(2,IC))
Number of third vertex (JC(3,IC))
Number of fourth vertex (JC(4,IC))
Number of the first side (JC(5,IC))
Number of the second side (JC(6,IC))

Number of the third side (JC(7,IC))

Number of the fourth side (JC(8,IC))

IV1 = JC(1,IC) first vertex

IV2 = JC(2,IC) second vertex

IV3 = JC(3,IC) third vertex

IV4 = JC(4,IC) fourth vertex

Seen from inside the tetrahedron, the first three vertices are counter clockwise around the large with the fourth vertex at the apex.

IS1 = JC(5,IC) first side

IS2 = JS(6,IC) second side

IS3 = JS(7,IC) third side

IS4 = JS(8,IC) fourth side

Face ISJ is opposite the IVJ vertex

11) A table of cells specifying:

x coordinate of the baricenter of cell	(XC(1,IC))
y coordinate of the baricenter of cell	(XC(2,IC))
z coordinate of the baricenter of cell	(XC(3,IC))
Volume of the cell	(XC(4,IC))

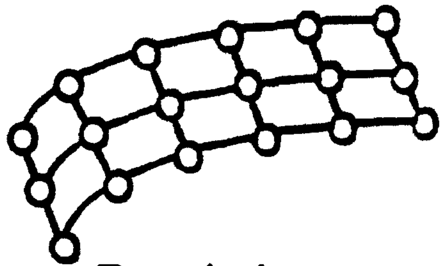
A.2.2 *Face(Triangle) information*

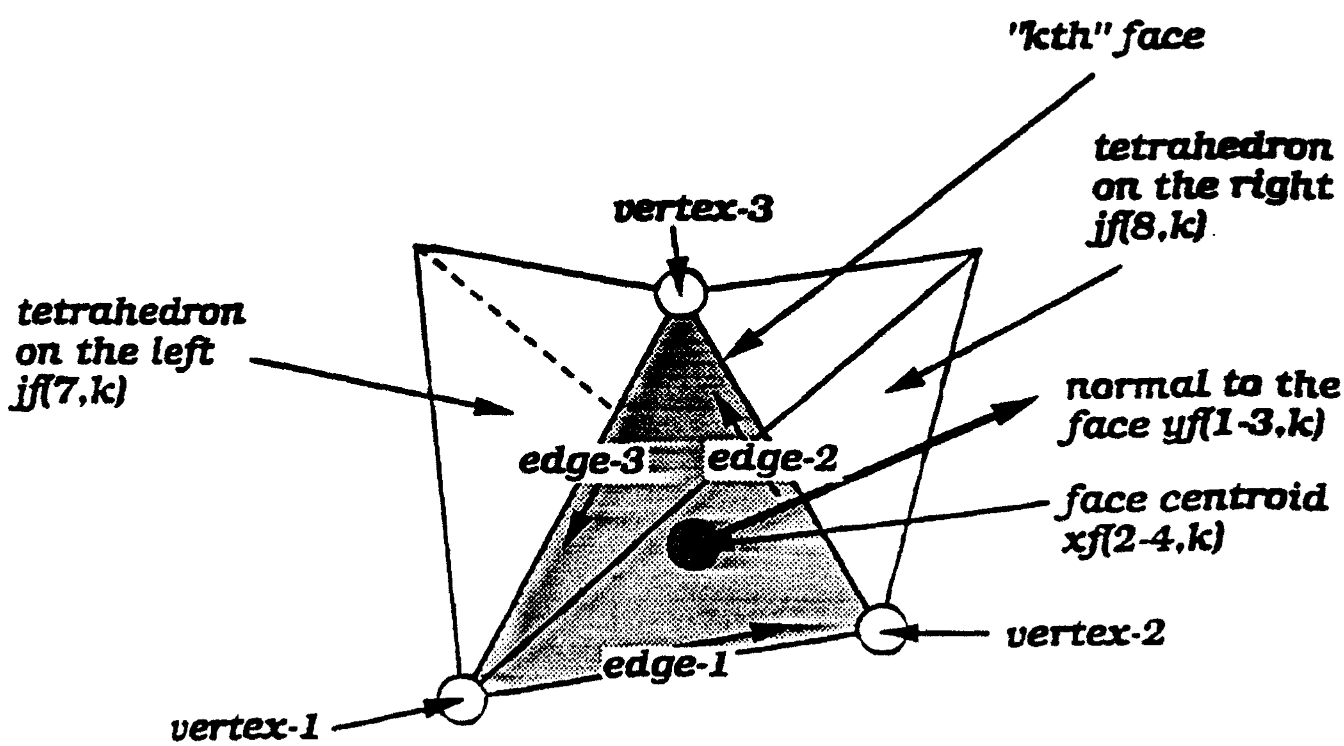
xf(1,k) - area of the kth face
xf(2,k) - x position of face centroid
xf(3,k) - y position of face centroid
xf(4,k) - z position of face centroid

yf(1,k) - x component of normal to face
yf(2,k) - y component of normal to face
yf(3,k) - z component of normal to face

jf(1,k) - the index of the first vertex
jf(2,k) - the index of the second vertex
jf(3,k) - the index of the third vertex
jf(4,k) - the signed index of the first edge
jf(5,k) - the signed index of the second edge
jf(6,k) - the signed index of the third edge
jf(7,k) - the index of the tetrahedron to the left of face
jf(8,k) - the index of the tetrahedron to the rigth of face
jf(9,k) - status of the kth face
- *s=0 face unrestricted*
- *s=1 not used*
- *s=2 face restricted to a surface*
- *s=3 face is fixed*

jf(10,k) - pointer to surface that restricts the face.
jf(11,k) - the boundary condition for this face

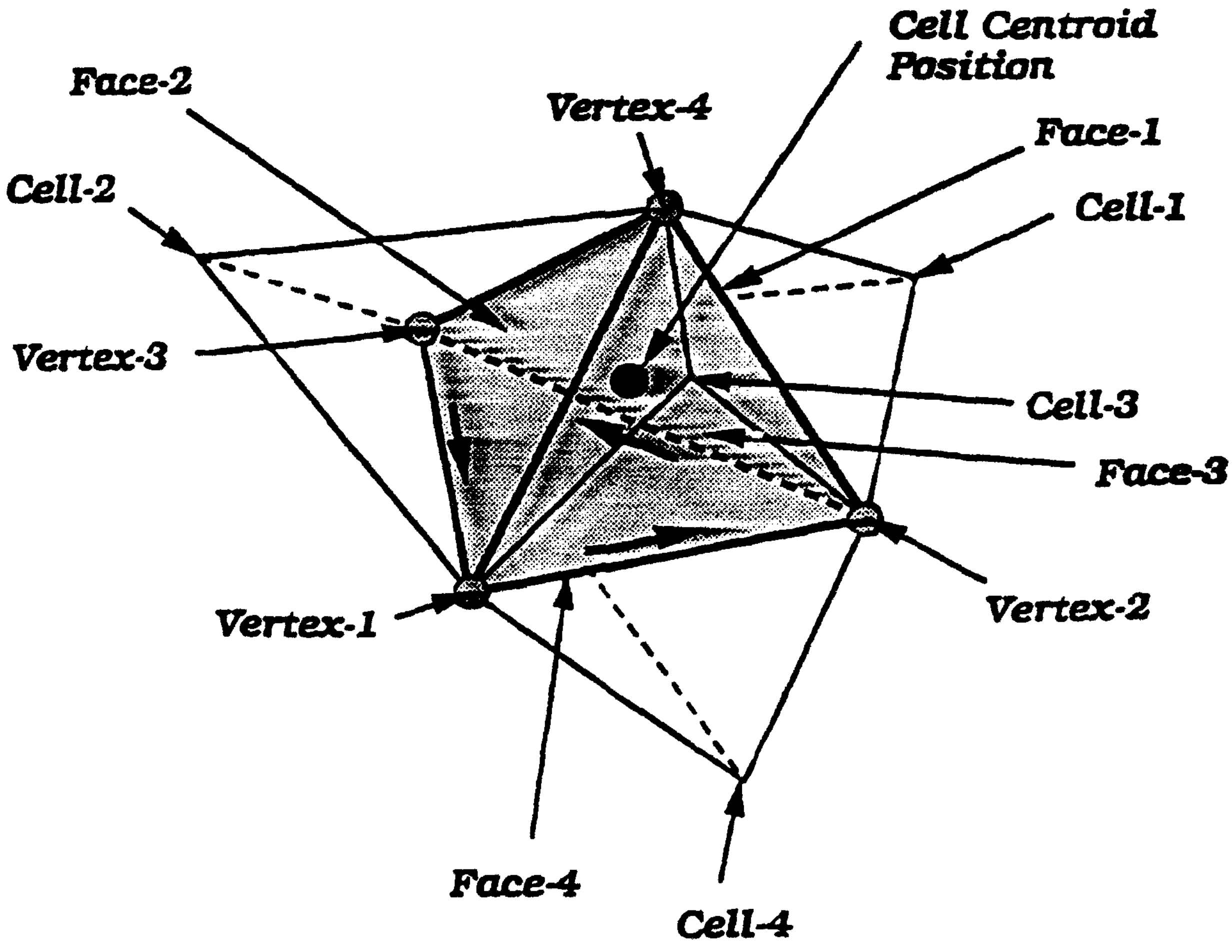




Cell(Tetrahedral) information

xc(1,k) - x position of cell centroid
xc(2,k) - y position of cell centroid
xc(3,k) - z position of cell centroid
xc(4,k) - volume of cell

jc(1,k) - the index of the first base vertex
jc(2,k) - the index of the second base vertex
jc(3,k) - the index of the third base vertex
jc(4,k) - the index of fourth vertex opposite base
jc(5,k) - the index of face opposite first vertex
jc(6,k) - the index of face opposite first vertex
jc(7,k) - the index of face opposite secondvertex
jc(8,k) - the index of face opposite third vertex
jc(9,k) - the index of face opposite fourth vertex
jc(10,k) - the index of cell opposite first vertex
jc(11,k) - the index of cell opposite second vertex
jc(12,k) - the index of cell opposite third vertex
jc(13,k) - the index of cell opposite fourth vertex



APPENDIX B

LISTINGS

Thu Jul 1 14:17:00 1993 threed.f Module List - alphabetical order

```
      PROGRAM AUGUSTT
C
C=====================================================================
C
C       The AUGUSTT Code
C
C                      -  Adaptive
C                      -  Unstructured
C                      -  Godunov
C                      -  Upwind
C                      -  Second order
C                      -  Triangular
C                      -  Three dimension
C
C                     The geometry structure comes from BERMUDA
C                     The solver is based on FUGGS
C
C
C       Version:   1.00      22 july, 1991
C
C       Authors:   Itzhak Lottati     (703)749-8648
C                  Shmuel Eidelman    (703)448-6491
C                  Adam Drobot        (703)734-5840
C
C                  Science Applications International Corporation
C                  Applied Physics Operation
C                  1710 Goodridge Drive
C                  McLean, Virginia 22102
C
C=====================================================================
C
C
C-----------------------------------------------------------------I
C                                                                 I
C     BERMUDA IS A MULTIDIMENSIONAL CODE WHICH IS BASED ON THE    I
C             USE OF TRIANGULAR GRIDS AS THE FUNDAMENTAL MESH     I
C             FOR FIELD LIKE QUANTITIES.  THE CODE REQUIRES       I
C             THAT ALL QUANTITIES ARE BASED AT THE BARICENTER     I
C             OF SIDES/TRIANGLES.                                 I
C                                                                 I
C             THE QUIP IS THAT THOSE WHO WORK ON BERMUDA          I
C             TRIANGLES ARE NEVER HEARD FROM AGAIN.               I
C                                                                 I
C             THE BASIC MODULES IN BERMUDA INCLUDE:               I
C                                                                 I
C                    A HYDRODYNAMICS CODE                         I
C                          .BASED ON A FIRST ORDER GODUNOV        I
C                           METHOD OR A SECOND ORDER GODUNOV      I
C                           WITH MESH ADAPTATION.                 I
C                                                                 I
C-----------------------------------------------------------------I
C
C     GRID SETUP TABLES AND THEIR MEANING:
C
C     +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C     +                                                         +
C     +   LIST OF VERTICES                                      +
C     +                                                         +
C     +        IV         - VERTEX INDEX                        +
C     +        XV(1,IV)   - X POSITION OF VERTEX                +
C     +        XV(2,IV)   - Y POSITION OF VERTEX                +
C     +        XV(3,IV)   - Z POSITION OF VERTEX                +
C     +                                                         +
C     +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C     +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C     +                                                         +
C     +   LIST OF EDGES                                         +
C     +                                                         +
C     +        IE         - EDGE INDEX                          +
C     +        JE(1,IE)   - INDEX OF LOWER EDGE VERTEX          +
C     +        JE(2,IE)   - INDEX OF UPPER EDGE VERTEX          +
C     +        JE(3,IE)   - INDEX OF LEFT  SIDE                 +
```

```
      C     +           JE(4,IE)    - INDEX OF RIGHT SIDE                    +
      C     +           XE(1,IE)    - LENGTH OF EDGE                         +
      C     +           XE(2,IE)    - DISTANCE BETWEEN ADJOINING SIDE        +
      C     +                         POINTS.                                +
      C     +                                                                +
      C     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      C
      C     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      C     +                                                                +
      C     +   LIST OF SIDES                                                +
      C     +                                                                +
      C     +        IS              - SIDE INDEX                            +
      C     +        JS(1,IS)    - INDEX OF FIRST  VERTEX                    +
      C     +        JS(2,IS)    - INDEX OF SECOND VERTEX                    +
      C     +        JS(3,IS)    - INDEX OF THIRD  VERTEX                    +
      C     +                                                                +
      C     +        THE VERTICES RUN AROUND THE SIDE IN ORDER               +
      C     +        COUNTER-CLOCKWISE FASHION                               +
      C     +                                                                +
      C     +        JS(4,IS)    - INDEX OF THE FIRST  EDGE                  +
      C     +        JS(5,IS)    - INDEX OF THE SECOND EDGE                  +
      C     +        JS(6,IS)    - INDEX OF THE THIRD  EDGE                  +
      C     +                                                                +
      C     +        THE EDGES ARE ARRANGED IN COUNTER-iCLOCKWISE            +
      C     +        FASHION. EDGE ONE RUNS FROM VERTEX-ONE TO               +
      C     +        VERTEX-TWO ETC.. THE SIGN OF JS(4-6,IS) INDICATES       +
      C     +        IF EDGE DATA IS STORED THE SAME WAY. IF IT IS           +
      C     +        JS>0 AND IT IS REVERSED JS<0                            +
      C     +        JS(7,IS)    - INDEX OF CELL ON LEFT                     +
      C     +        JS(8,IS)    - INDEX OF CELL ON RIGHT                    +
      C     +                                                                +
      C     +        XS(1,IS)      - X POSITION OF CENTROID OF TRIANGLE      +
      C     +        XS(2,IS)      - Y POSITION OF CENTROID OF TRIANGLE      +
      C     +        XS(3,IS)      - Z POSITION OF CENTROID OF TRIANGLE      +
      C     +        XS(4,IS)      - AREA OF TRIANGLE                        +
      C     +        XS(5,IS)      - DISTANCE BETWEEN ADJOINING CELLS        +
      C     +                        POINTS CROSSING TRIANGLE IS             +
      C     +                                                                +
      C     +                                                                +
      C     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      C
      C     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      C     +                                                                +
      C     +   LIST OF CELLS                                                +
      C     +                                                                +
      C     +        IC              - CELL INDEX                            +
      C     +        JC(1,IC)    - INDEX OF FIRST  VERTEX                    +
      C     +        JC(2,IC)    - INDEX OF SECOND VERTEX                    +
      C     +        JC(3,IC)    - INDEX OF THIRD  VERTEX                    +
      C     +        JC(4,IC)    - INDEX OF FOURTH VERTEX                    +
      C     +                                                                +
      C     +        THE CONVENTION FOR VERTICES IS THAT 1-3                 +
      C     +        ARE ARRANGED COUNTER-CLOCKWISE ABOUT THE                +
      C     +        BASE AND THAT 4 IS AT THE APEX.                         +
      C     +                                                                +
      C     +        JC(5,IC)    - INDEX OF FIRST  SIDE                      +
      C     +        JC(6,IC)    - INDEX OF SECOND SIDE                      +
      C     +        JC(7,IC)    - INDEX OF THIRD  SIDE                      +
      C     +        JC(8,IC)    - INDEX OF FOURTH SIDE                      +
      C     +                                                                +
      C     +        THE CONVENTION FOR SIDES IS THAT SIDE ONE COVERS        +
      C     +        THE SPACE BETWEEN VERTEX-ONE,VERTEX-TWO,AND THE         +
      C     +        VERTEX AT THE APEX ETC.. SIDE FOUR IS THE BASE          +
      C     +                                                                +
      C     +        XC(1,IC)    - X POSITION OF CELL POINT                  +
      C     +        XC(2,IC)    - Y POSITION OF CELL POINT                  +
      C     +        XC(3,IC)    - Z POSITION OF CELL POINT                  +
      C     +        XC(4,IC)    - CELL VOLUME.                              +
      C     +                                                                +
      C     +                                                                +
      C     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      C
      C
      C=====================================================================
```

```
C
C --- DEFINITION FOR ALL HYDRODYNAMIC QUANTITIES ---------------------
C
C-------------------------------------------------------------I
C                                                             I
C     USE OF PARAMETERS:                                      I
C                                                             I
C         MHQ     - MAXIMUM NUMBER OF HYDRO QUANTITIES.       I
C                                                             I
C                                                             I
C-------------------------------------------------------------I
C
C====================================================================
C
      include     'dmsh00.h'
      include     'dhydm0.h'
      include     'dphsm0.h'
      include     'dmtrl0.h'
C
      REAL XX(600),PP(600),HR(600),
     .     UU(600),GG(600),AA(600),EE(600)
      DOUBLE PRECISION VOL1,VOL2,VOL3,VOL4,VOLL,XXI,YYI,ZZI
      DOUBLE PRECISION DEFVOL
      OPEN(2 ,FILE='data.dd',FORM='FORMATTED')
      OPEN(4 ,FILE='thermo.d',FORM='FORMATTED')
      OPEN(8 ,FILE='threed2.5',FORM='UNFORMATTED')
      OPEN(88 ,FILE='threed82',FORM='UNFORMATTED')
      OPEN(9 ,FILE='threed3',FORM='UNFORMATTED')
      OPEN(10,FILE='threed4',FORM='FORMATTED')
      OPEN(15,FILE='AVSfmhall.inp',FORM='FORMATTED')
      OPEN(14,FILE='AVSsmhall.inp',FORM='FORMATTED')
      OPEN(16,FILE='OUTPUT.MSH',FORM='FORMATTED')
      OPEN(26,FILE='EXPLSV.RND',FORM='FORMATTED')
      OPEN(17,FILE='ve0640.stv',FORM='FORMATTED')
      OPEN(18,FILE='f0640.stv',FORM='FORMATTED')
      OPEN(19,FILE='pr640.stv',FORM='FORMATTED')
      OPEN(11,FILE='truck.input.8b',STATUS='OLD'      )
C
C====================================================================
C
C      NAMELIST /DATA/ ICOND,ICONP,IOPTN,XMCHIN,RIN,PIN,ALFA,HRGG,IHRN,
C     .                NTIME,NDUMP,IOPORD
C
C-------------------------------------------------------------------I
C                                                                   I
C                                                                   I
C     --- MEANING OF NAMELIST VARIABLES:                            I
C                                                                   I
C     ICOND = 0 READ INPUT GRID FOR A NEW RUN                       I
C           = 1 READ THE GRID FROM PREVIOUS RUN                     I
C     ICONP = 0 PRIMITIVE VARIABLES SET TO ZERO                     I
C           = 1 VARIABLES READ FROM PREVIOUS RUN                    I
C     IOPTN = 1 SOLUTION FOR STEADY STATE,                          I
C           = 2 SOLUTION FOR TRANSIENT PHENOMENA                    I
C                                                                   I
C     XMCHIN = FOR TRANSIENT SHOCK CALCULATIONS(IOPTN=2)THIS VARIABLE I
C              IS USED TO SPECIFY THE UPSTREAM MACH NUMBER          I
C                                                                   I
C              RIN = THE AMBIENT DENSITY IN THE CHAMBER             I
C                                                                   I
C              PIN = THE AMBIENT PRESSURE IN THE CHAMBER            I
C                                                                   I
C              APPLYING NORMAL SHOCK WAVES RELATIONS FOR AN ADIABATIC I
C              FLOW RELATION STATIC-PRESSURE RATIO ACROSS THE SHOCK I
C              AS WELL AS THE DENSITY RATIO AND MACH NUMBER RATIO   I
C              ARE COMPUTED TO SET CORRECTLY THE CONDITION AT THE   I
C              INLET EDGES( EDGE BOUNDARY 8 ) OF THE COMPUTATIONAL  I
C              DOMAIN                                               I
C                                                                   I
C              FOR STEADY STATE SHOCK CALCULATIONS(IOPTN=1)THIS IS THE I
C              INFLOW MACH NUMBER. ALL DOMAIN VELOCITIES ARE THEN   I
C              INITIALIZED WITH THIS VALUE.                         I
C                                                                   I
C              RIN = THE AMBIENT DENSITY AT INFINITY                I
```

```
      C                                                                   I
      C                PIN = THE AMBIENT PRESSURE AT INFINITY             I
      C                                                                   I
      C                ALL COMPUTATIONAL DOMAIN ARE THEN INITIALIZED WITH I
      C                THOSE VALUES.                                      I
      C                                                                   I
      C      ALFA  = THE DIRECTION OF INFLOW IN DEGREES RELATIVE TO A RIGHT I
      C              HAND COORDINATE SYSTEM. ALFA=0 MEANS FLOW FROM LEFT TO I
      C              RIGHT. ALFA=90 MEANS FROM BOTTOM TO TOP. ALFA=-90 OR 270 I
      C              MEANS FLOW FROM TOP TO BOTTOM ETC.                   I
      C      HRGG  = INITIAL GAMMA IN THE EQUATION OF STATE              I
      C              THE CODE RUNS USING THE AIR EQUATION AS A BASELINE AND I
      C              SHOULD BE MODIFIED IF SOMETHING ELSE IS DESIRED.     I
      C      IHRN  = NUMBER OF ITERATIONS IN THE RIEMANN SOLVER TO FIND THE I
      C              DIAPHRAGM SOLUTION.(3 to 4 SHOULD BE USED AND INCREASED I
      C              ONLY FOR HIGH MACH NUMBER CASES).                    I
      C                                                                   I
      C      NTIME = NUMBER OF REPEATS FOR THE INTEGRATION SEQUENCE.     I
      C              AN OUTPUT DUMP IS DONE EVERY SEQUENCE REPEAT.        I
      C      NDUMP = NUMBER OF ITERATIONS IN THE INNER LOOP              I
      C                                                                   I
      C         +_____________________ o NTIME - DUMPING DATA             I
      C         I                                                         I
      C         I              +______ o NDUMP - INTEGRATION              I
      C         I              I                                          I
      C         I              I                                          I
      C         I              +______ o ..... INNER LOOP                 I
      C         I                                                         I
      C         +_____________________ o ..... DUMPING LOOP               I
      C                                                                   I
      C      IOPORD = 1 THE CODE WILL RUN FIRST ORDER GODUNOV METHOD      I
      C             = 2 THE CODE WILL RUN SECOND ORDER GODUNOV METHOD     I
      C                                                                   I
      C-------------------------------------------------------------------I
      C
             ICOND = 0
             ICONP = 0
             IOPTN = 1
             IEOS  = 1
      C
             XMCHIN = 2.5
             RIN    = 1.25
             PIN    = 101350.
             RGAS   = 8314.3
             GPERCC = .001
      C
             ALFA = 0.
             HRGG = 1.4
             IHRN = 4
             NTIME = 12
             NDUMP = 200
             IOPORD = 2
      C
      C --- READ THE INPUT DATA -------------------------------------------
      C
               READ (2,DATA)
      C
      C --- PRINTOUT THE RUN PARAMETERS -----------------------------------
      C
              PRINT 101,    ICOND,ICONP,IOPTN,XMCHIN,RIN,PIN,ALFA,HRGG,IHRN,
           .                 NTIME,NDUMP,IOPORD
      C
      C --- SET RUN CONDITIONS AND PRINTOUT TO CONSOLE ---------------------
      C
      C      READING GRID DATA FROM EDGE.ZON
      C
              THIRD = 1. / 3.
               IF( ICOND . EQ . 0 ) THEN
      C
               CALL UPDATE
      C
              ELSE
      C
               CALL UPGRAD
```

```
C
         END IF
         CALL MATRLA
         CALL MATRLX
C
C --- INITIALIZATION OF THE PROBLEM
C
            HRSM = 1.E-8
            HRGP = HRGG + 1.
            HRGM = HRGG - 1.
            CF   = HRGP / ( 2. * HRGG )
        TT = 0.
C
        PIRAD = ATAN( 1. ) / 45.
        ALPHA = ALFA * PIRAD
        PRINT *,ALFA,PIRAD,ALPHA
C
        COSS = COS( ALPHA )
        SINN = SIN( ALPHA )
        TANN = TAN( ALPHA )
C
C===================================================================
C
C --- SET THE INITIAL VALUE FOR PRIMITIVE VARIABLES -------------------
C
C(2)>>>>
         TLIMIT = .9
         ITER = 6
        IF( ICOND . EQ . 0 ) THEN
          UVIN = XMCHIN * SQRT( HRGG * PIN / RIN )
          UIN = UVIN * COSS
          VIN = UVIN * SINN
          WIN = 0.
C
          DO 150 IC = 1 , NC
             HYDV( IC , 1 ) = RIN
             HYDV( IC , 2 ) = 0.
             HYDV( IC , 3 ) = 0.
             HYDV( IC , 4 ) = 0.
             HYDV( IC , 5 ) = PIN
             HYDV( IC , 6 ) = 1.E-6
             HYDV( IC , 7 ) = 1.4
             HYDV( IC , 8 ) = PIN / ( HYDV( IC , 7 ) - 1. )
C
 150      CONTINUE
          RADIUS = .0001
          EXPLSV = 8.
          DO IC = 1 , NC
             XXI = XC( 1 , IC )
             YYI = XC( 2 , IC )
             ZZI = XC( 3 , IC )
             RSS = SQRT( XXI * XXI + YYI * YYI + ZZI * ZZI )
             IF( RSS . LT . RADIUS ) THEN
               print*,xxi,yyi,zzi,radius
                HYDV( IC , 1 ) = EXPLSV * .4536 * .75 / 3.141569 /
     .                           ( RADIUS * RADIUS * RADIUS )
                HYDV( IC , 6 ) = 1.
                HYDV( IC , 8 ) = HYDV( IC , 1 ) * 1080. * 4.185 *
     .                                      1000. * 1.01 / .7
       NITER = 0
       DST = HYDV( IC , 1 ) * GPERCC
       VOL = WMX * ( 1. - DST / FSX ) / DST / XGX
       EMEO = HYDV( IC , 8 ) / HYDV( IC , 1 ) * WMX / RGAS
C
       IYY = ( EMEO - EMEOX( 3 ) ) / RANGEX + 1
       IYY = MAXO( 1 , MINO( IYY , 47 ) )
C
       K = IYY + 2
       IYY = IYY
     . + INT( AMAX1( EMEO - EMEOX( K ) , 0.) / DYX( K ) )
     . - INT( AMAX1( EMEOX( K + 1 ) - EMEO , 0. ) / DYX( K ) )
       IYY = MAXO( 1, MINO( IYY , 47 ) )
C
       K1 = IYY + 2
```

```
          K2 = K1 + 1
          RT = ( EMEO - EMEOX( K1 ) ) / ( EMEOX( K2 ) - EMEOX( K1 ) )
          T = TX( K1 ) + 100. * RT
          CVM = CVMX( K1 ) + RT * ( CVMX( K2 ) - CVMX( K1 ) )
          ERS = 0.
C
 10       CONTINUE
          P = RGAS * T / VOL / GPERCC
          RGAMM1 = CVM
C
          X = COVX / VOL / ( ( T + THETAX ) ** ALFAX )
          Z = X * EXP( BETAX * X )
          X = 1. + BETAX * X
          RT = ALFAX * T / ( T + THETAX )
          ERS = ERS + RT * Z * T
C
          IF ( ITER .EQ. NITER ) GO TO 20
C
          CVM = CVM * XGX + SCVX
     *                + RT * Z * ( 2. - RT / ALFAX - RT * X )
          T = T - AMIN1( ERS / CVM , TLIMIT * T )
C
          NITER = NITER + 1
C
          RT = 0.01 * T
          K1 = RT
          K1 = MIN0 ( K1, 49 )
          K1 = MAX0 ( K1, 3  )
          K2 = K1 + 1
          RT = RT - K1
          CVM = CVMX(K1) + RT * ( CVMX( K2 ) - CVMX( K1 ) )
          ERS = EMEOX( K1 ) + RT * ( EMEOX( K2 ) - EMEOX( K1 ) )
          ERS = ERS - EMEO
C
          GO TO 10
C
 20       CONTINUE
          P = P * ( 1. + Z )
          RGAMM1 = ( RGAMM1 +
     *              RT * Z * ( 2. - RT / ALFAX - RT * X ) ) / ( 1. + Z )
          X = X * Z / ( 1. + Z )
          RGAMM1 = RGAMM1 / ( ( 1. - RT * X ) ** 2 + X * RGAMM1 )
          ERS = ERS / EMEO
          HYDV( IC , 7 ) = 1. / RGAMM1 + 1.
          HYDV( IC , 5 ) = P
          END IF
          END DO
C
          XCOUNT = 0
          DO IC = 1 , NC
          RCOUNT = HYDV( IC , 8 ) + .5 * HYDV( IC , 1 ) *
     .                      ( HYDV( IC , 2 ) * HYDV( IC , 2 ) +
     .                        HYDV( IC , 3 ) * HYDV( IC , 3 ) +
     .                        HYDV( IC , 4 ) * HYDV( IC , 4 ) )
          XCOUNT = XCOUNT + XC( 4 , IC ) * RCOUNT
          END DO
           PRINT * ,XCOUNT
              IIJJ=1
              IF(IIJJ.EQ.0) GO TO 1122
C        remove the followed IF statement for regular run
C          IF( IOPTN . EQ . 2 ) THEN
C          IF( IOPTN . EQ . 1 ) THEN
C
              NX = 360
              DO 190 IX = 1 , NX
                 XX(IX) = (IX-.5)*.002
 190          CONTINUE
              READ (11,1001) (PP(IX),IX=1,NX)
              READ (11,1001) (UU(IX),IX=1,NX)
              READ (11,1001) (HR(IX),IX=1,NX)
              READ (11,1001) (AA(IX),IX=1,NX)
              READ (11,1001) (GG(IX),IX=1,NX)
              READ (11,1001) (EE(IX),IX=1,NX)
 1001     FORMAT(6E12.5)
```

```
C
          ICOUNT = 0
          DO 260 IC = 1 , NC
C            XXI = XC( 1 , IC ) + .2667
           XXI = XC( 1 , IC ) + .1143
           YYI = XC( 2 , IC ) - 1.96596
C           ZZI = XC( 3 , IC ) - 1.25
           ZZI = XC( 3 , IC ) - 1.905
           RSS = SQRT( XXI * XXI + YYI * YYI + ZZI * ZZI )
           XYS = SQRT( XXI * XXI + YYI * YYI )
C
           DO 270 IX = 1 , NX-1
              XDD1 = XX( IX )
              XDD2 = XX( IX+1 )
              IF( RSS . GT . XDD1 . AND . RSS . LT . XDD2 ) THEN
                 XKSI = ( RSS - XDD1 ) / ( XDD2 - XDD1 )
                 ICOUNT = ICOUNT + 1
C
                 HYDV(IC,1) = HR(IX) * (1.-XKSI) +
     .                            HR(IX+1) * XKSI
C
                 HYDUVW = UU(IX) * (1.-XKSI) +
     .                         UU(IX+1) * XKSI
                 HYDV(IC,4) = ZZI / RSS * HYDUVW
                 HYDVUV = XYS / RSS * HYDUVW
C
                 THETHA = ATAN2( YYI , XXI )
                 HYDV(IC,2) = HYDVUV * COS( THETHA )
                 HYDV(IC,3) = HYDVUV * SIN( THETHA )
C
                 HYDV(IC,5) = PP(IX) * (1.-XKSI) +
     .                            PP(IX+1) * XKSI
C                HYDV(IC,5) = 1.08*HYDV(IC,5)
                 HYDV(IC,7) = GG(IX) * (1.-XKSI) +
     .                            GG(IX+1) * XKSI
                 HYDV(IC,6) = AA(IX) * (1.-XKSI) +
     .                            AA(IX+1) * XKSI
                 HYDV(IC,8) = EE(IX) * (1.-XKSI) +
     .                            EE(IX+1) * XKSI
C
                 GOTO 301
              ENDIF
 270       CONTINUE
 301       CONTINUE
      NITER = 6
      IF(NITER.EQ.0) THEN
      IF( HYDV( IC , 6 ) . LT . .2 ) THEN
      DST = HYDV( IC , 1 ) * GPERCC
      VOL = WMA * ( 1. - DST / FSA ) / DST / XGA
      TT = HYDV( IC , 5 ) * VOL * GPERCC / RGAS
C
      T = TT
      RT = 0.01 * T
      K1 = RT
      K1 = MIN0 ( K1, 49 )
      K1 = MAX0 ( K1, 3  )
      K2 = K1 + 1
      RT = RT - K1
      ENERGY = EMEOA( K1 ) + RT * ( EMEOA( K2 ) - EMEOA( K1 ) )
      ENERGY = ENERGY * RGAS / WMA
C
      DO ITER = 1 , NITER
      X = COVA / VOL / ( T + THETAA ) ** ALFAA
C
      BETAZX = BETAA * X
      RT = X * EXP( BETAZX )
      RTINV = 1. / ( 1. + RT )
C -   ERS IS THE FUNCTION, RT IS THE DERIVATIVE
      ERS = T -  TT * RTINV
      RT = 1. - TT * RTINV * RTINV * ALFAA * RT * ( 1. + BETAZX ) /
     .           ( T + THETAA )
      ERS = ERS / RT
      T = T - ERS
      END DO
```

```
C
      RT = 0.01 * T
      K1 = RT
      K1 = MIN0 ( K1, 49 )
      K1 = MAX0 ( K1, 3  )
      K2 = K1 + 1
      RT = RT - K1
      ENERGY = EMEOA( K1 ) + RT * ( EMEOA( K2 ) - EMEOA( K1 ) )
C
      X = COVA / VOL / ( ( T + THETAA ) ** ALFAA )
      EX = EXP( BETAA * X )
      Z = X * EX
      RT = ALFAA * T / ( T + THETAA )
      ENERGY = ENERGY + RT * Z * T
      HYDV( IC , 8 ) = ENERGY * RGAS / WMA
      EMEO = HYDV( IC , 8 ) / HYDV( IC , 1 ) * WMA / RGAS
C
      IYY = ( EMEO - EMEOA( 3 ) ) / RANGEA + 1
      IYY = MAX0( 1 , MIN0( IYY , 47 ) )
C
      K = IYY + 2
      IYY = IYY
     . + INT( AMAX1( EMEO - EMEOA( K ) , 0.) / DYA( K ) )
     . - INT( AMAX1( EMEOA( K + 1 ) - EMEO , 0. ) / DYA( K ) )
      IYY = MAX0( 1, MIN0( IYY , 47 ) )
C
      K1 = IYY + 2
      K2 = K1 + 1
      RT = ( EMEO - EMEOA( K1 ) ) / ( EMEOA( K2 ) - EMEOA( K1 ) )
      T = TA( K1 ) + 100. * RT
      CVM = CVMA( K1 ) + RT * ( CVMA( K2 ) - CVMA( K1 ) )
      ERS = 0.
C
      P = RGAS * T / VOL / GPERCC
      RGAMM1 = CVM
      HYDV( IC , 7 ) = 1. / RGAMM1 + 1.
      HYDV( IC , 5 ) = P
C
      ELSE
C
      DST = HYDV( IC , 1 ) * GPERCC
      VOL = WMX * ( 1. - DST / FSX ) / DST / XGX
      TT = HYDV( IC , 5 ) * VOL * GPERCC / RGAS
C
      T = TT
      RT = 0.01 * T
      K1 = RT
      K1 = MIN0 ( K1, 49 )
      K1 = MAX0 ( K1, 3  )
      K2 = K1 + 1
      RT = RT - K1
      ENERGY = EMEOX( K1 ) + RT * ( EMEOX( K2 ) - EMEOX( K1 ) )
      ENERGY = ENERGY * RGAS / WMX
C
      DO ITER = 1 , NITER
      X = COVX / VOL / ( T + THETAX ) ** ALFAX
C
      BETAZX = BETAX * X
      RT = X * EXP( BETAZX )
      RTINV = 1. / ( 1. + RT )
C -   ERS IS THE FUNCTION, RT IS THE DERIVATIVE
      ERS = T -  TT * RTINV
      RT = 1. - TT * RTINV * RTINV * ALFAX * RT * ( 1. + BETAZX ) /
     .                ( T + THETAX )
      ERS = ERS / RT
      T = T - ERS
      END DO
C
      RT = 0.01 * T
      K1 = RT
      K1 = MIN0 ( K1, 49 )
      K1 = MAX0 ( K1, 3  )
      K2 = K1 + 1
      RT = RT - K1
```

```
      ENERGY = EMEOX( K1 ) + RT * ( EMEOX( K2 ) - EMEOX( K1 ) )
C
      X = COVX / VOL / ( ( T + THETAX ) ** ALFAX )
      EX = EXP( BETAX * X )
      Z = X * EX
      RT = ALFAX * T / ( T + THETAX )
      ENERGY = ENERGY + RT * Z * T
      HYDV( IC , 8 ) = ENERGY * RGAS / WMX
      VOL = WMX * ( 1. - DST / FSX ) / DST / XGX
      EMEO = HYDV( IC , 8 ) / HYDV( IC , 1 ) * WMX / RGAS
C
      IYY = ( EMEO - EMEOX( 3 ) ) / RANGEX + 1
      IYY = MAX0( 1 , MIN0( IYY , 47 ) )
C
      K = IYY + 2
      IYY = IYY
     . + INT( AMAX1( EMEO - EMEOX( K ) , 0.) / DYX( K ) )
     . - INT( AMAX1( EMEOX( K + 1 ) - EMEO , 0. ) / DYX( K ) )
      IYY = MAX0( 1, MIN0( IYY , 47 ) )
C
      K1 = IYY + 2
      K2 = K1 + 1
      RT = ( EMEO - EMEOX( K1 ) ) / ( EMEOX( K2 ) - EMEOX( K1 ) )
      T = TX( K1 ) + 100. * RT
      CVM = CVMX( K1 ) + RT * ( CVMX( K2 ) - CVMX( K1 ) )
      ERS = 0.
C
 401  CONTINUE
      P = RGAS * T / VOL / GPERCC
      RGAMM1 = CVM
C
      X = COVX / VOL / ( ( T + THETAX ) ** ALFAX )
      Z = X * EXP( BETAX * X )
      X = 1. + BETAX * X
      RT = ALFAX * T / ( T + THETAX )
      ERS = ERS + RT * Z * T
C
      IF ( ITER .EQ. NITER ) GO TO 201
C
      CVM = CVM * XGX + SCVX
     *              + RT * Z * ( 2. - RT / ALFAX - RT * X )
      T = T - AMIN1( ERS / CVM , TLIMIT * T )
C
      NITER = NITER + 1
C
      RT = 0.01 * T
      K1 = RT
      K1 = MIN0 ( K1, 49 )
      K1 = MAX0 ( K1, 3  )
      K2 = K1 + 1
      RT = RT - K1
      CVM = CVMX(K1) + RT * ( CVMX( K2 ) - CVMX( K1 ) )
      ERS = EMEOX( K1 ) + RT * ( EMEOX( K2 ) - EMEOX( K1 ) )
      ERS = ERS - EMEO
C
      GO TO 401
C
 201  CONTINUE
      P = P * ( 1. + Z )
      RGAMM1 = ( RGAMM1 +
     *            RT * Z * ( 2. - RT / ALFAX - RT * X ) ) / ( 1. + Z )
      X = X * Z / ( 1. + Z )
      RGAMM1 = RGAMM1 / ( ( 1. - RT * X ) ** 2 + X * RGAMM1 )
      ERS = ERS / EMEO
      HYDV( IC , 7 ) = 1. / RGAMM1 + 1.
      HYDV( IC , 5 ) = P
      END IF
      END IF
 260     CONTINUE
C
C(2)----
      ELSE
C
         XMSQR = XMCHIN * XMCHIN
```

```
            PINL = PIN
            RINL = RIN
            RINRTO = ( HRGG + 1. ) * XMSQR /
     .               ( ( HRGG - 1. ) * XMSQR + 2. )
            PINRTO = ( 2. * HRGG * XMSQR - ( HRGG - 1. ) ) /
     .                                          ( HRGG + 1. )
            PIN = PINRTO * PINL
            RIN = RINRTO * RINL
            YMCHIN = SQRT( ( ( HRGG - 1. ) * XMSQR + 2. ) /
     .                 ( 2. * HRGG * XMSQR - ( HRGG - 1. ) ) )
            PRINT*,HRGG,RIN,PIN,YMCHIN
            PRINT*,HRGG,RINL,PINL,XMCHIN
            UVIN = XMCHIN * SQRT( HRGG * PINL / RINL ) -
     .               YMCHIN * SQRT( HRGG * PIN / RIN )
            UIN = UVIN * COSS
            VIN = UVIN * SINN
            WIN = 0.
C
            DO 155 IC = 1 , NC
               HYDV( IC , 1 ) = RINL
               HYDV( IC , 2 ) = UIN
               HYDV( IC , 3 ) = VIN
               HYDV( IC , 4 ) = WIN
               HYDV( IC , 5 ) = PINL
C
 155        CONTINUE
         ENDIF
C        remove the followed END IF for regular run
C          ENDIF
C          ENDIF
C(2)<<<<
C
1122     CONTINUE
         IF( ICOND . EQ . 0 ) THEN
         NPRTCL = 25
         XPRTCL(1,1) = .443
         XPRTCL(2,1) = 1.0414
         XPRTCL(3,1) = 1.4224
         XPRTCL(1,2) = -.002
         XPRTCL(2,2) = .3556
         XPRTCL(3,2) = 0.5842
         XPRTCL(1,3) = -.275
         XPRTCL(2,3) = -.3058
         XPRTCL(3,3) = -1.4224
         XPRTCL(1,4) = 2.032
         XPRTCL(2,4) = -.3048
         XPRTCL(3,4) = -4.572
         XPRTCL(1,5) = .3048
         XPRTCL(2,5) = .1016
         XPRTCL(3,5) = .3048
         XPRTCL(1,6) = .4572
         XPRTCL(2,6) = .1016
         XPRTCL(3,6) = .4572
         XPRTCL(1,7) = .6096
         XPRTCL(2,7) = .1016
         XPRTCL(3,7) = .3048
         XPRTCL(1,8) = .4572
         XPRTCL(2,8) = .1016
         XPRTCL(3,8) = .1524
         XPRTCL(1,9) = 1.3462
         XPRTCL(2,9) = .1016
         XPRTCL(3,9) = .3048
         XPRTCL(1,10) = 1.4986
         XPRTCL(2,10) = .1016
         XPRTCL(3,10) = .4572
         XPRTCL(1,11) = 1.651
         XPRTCL(2,11) = .1016
         XPRTCL(3,11) = .3048
         XPRTCL(1,12) = 1.4986
         XPRTCL(2,12) = .1016
         XPRTCL(3,12) = .1524
         XPRTCL(1,13) = .6096
         XPRTCL(2,13) = .7740
         XPRTCL(3,13) = 1.0668
```

```
      XPRTCL(1,14) = .6096
      XPRTCL(2,14) = .8138
      XPRTCL(3,14) = .5334
      XPRTCL(1,15) = 1.4224
      XPRTCL(2,15) = .7740
      XPRTCL(3,15) = 1.0668
      XPRTCL(1,16) = 1.4224
      XPRTCL(2,16) = .8128
      XPRTCL(3,16) = .5334
      XPRTCL(1,17) = -.3058
      XPRTCL(2,17) = 1.3208
      XPRTCL(3,17) = -.4318
      XPRTCL(1,18) = .2032
      XPRTCL(2,18) = .7590
      XPRTCL(3,18) = 1.1898
      XPRTCL(1,19) = .254
      XPRTCL(2,19) = .1772
      XPRTCL(3,19) = 1.1948
      XPRTCL(1,20) = .9144
      XPRTCL(2,20) = .4064
      XPRTCL(3,20) = .9652
      XPRTCL(1,21) = .2032
      XPRTCL(2,21) = .7680
      XPRTCL(3,21) = 1.1888
      XPRTCL(1,22) = .1532
      XPRTCL(2,22) = .7670
      XPRTCL(3,22) = 1.1888
      XPRTCL(1,23) = .1532
      XPRTCL(2,23) = .7665
      XPRTCL(3,23) = 1.1878
      XPRTCL(1,24) = .1532
      XPRTCL(2,24) = .7765
      XPRTCL(3,24) = 1.1898
      XPRTCL(1,25) = .1532
      XPRTCL(2,25) = .7655
      XPRTCL(3,25) = 1.1898
      DO IK = 1 , NPRTCL
      RMINN = 100000000.
      DO IC = 1 , NC
      I1=JC(1,IC)
      I2=JC(2,IC)
      I3=JC(3,IC)
      I4=JC(4,IC)
      XXI = XPRTCL( 1 , IK )
      YYI = XPRTCL( 2 , IK )
      ZZI = XPRTCL( 3 , IK )
      CALL VOLMTETC ( I1, I2, I3, XXI, YYI, ZZI , VOL1 )
      CALL VOLMTETC ( I1, I2, I4, XXI, YYI, ZZI , VOL2 )
      CALL VOLMTETC ( I1, I3, I4, XXI, YYI, ZZI , VOL3 )
      CALL VOLMTETC ( I2, I3, I4, XXI, YYI, ZZI , VOL4 )
      XXI = XV( 1 , I4 )
      YYI = XV( 2 , I4 )
      ZZI = XV( 3 , I4 )
      CALL VOLMTETC ( I1, I2, I3, XXI, YYI, ZZI , VOLL )
      DEFVOL=DABS(VOL1)+DABS(VOL2)+DABS(VOL3)+DABS(VOL4)
     .      -DABS(VOLL)
      IF( DABS(DEFVOL/VOLL) . LT . .001 ) THEN
          IJKPRT(IK) = IC
          PRINT*,ik,vol1,vol2,vol3,vol4
          PRINT*,ic,voll,defvol,defvol/voll
          PRINT*,(XV(kk,jc(1,ic)),kk=1,3)
          PRINT*,(XV(kk,jc(2,ic)),kk=1,3)
          PRINT*,(XV(kk,jc(3,ic)),kk=1,3)
          PRINT*,(XV(kk,jc(4,ic)),kk=1,3)
          PRINT*,(JS(9,jc(kk,ic)),kk=5,8)
      END IF
      END DO
      END DO
      DO IK = 1 , NPRTCL
          IC = IJKPRT(IK)
          ISS = JC(5,IC)
      DO IKK = 5 , 8
          IS = JC(IKK,IC)
          IBC = JS(9,IS)
```

```
               IF( IBC . EQ . 6 ) THEN
                ISS = IS
               END IF
            END DO
                IJKPRT(IK) = ISS
            END DO
            END IF
      C
             PRINT * ,ICOND,ICONP
      C
             IF( ICONP . EQ . 1 ) THEN
             READ(8) RIN,PIN,RINL,PINL,UVIN,UIN,VIN,WIN,TT
             PRINT *, RIN,PIN,RINL,PINL,UVIN,UIN,VIN,WIN,TT
      C        READ (8) NPRTCL
      C        IF(NPRTCL.GT.0)
      C     .  READ (8) (IJKPRT(IK),IK=1,NPRTCL)
             DO II = 1 , 5
             READ(8) ((HYDV(IC,IK),IK=1,8),IC=1,NC)
             END DO
      C
            END IF
      C
            ZCOUNT = 0
            DO 380 IC = 1 , NC
            RCOUNT = HYDV( IC , 8 ) + .5 * HYDV( IC , 1 ) *
           .                     ( HYDV( IC , 2 ) * HYDV( IC , 2 ) +
           .                       HYDV( IC , 3 ) * HYDV( IC , 3 ) +
           .                       HYDV( IC , 4 ) * HYDV( IC , 4 ) )
            ZCOUNT = ZCOUNT + XC( 4 , IC ) * RCOUNT
        380 CONTINUE
             YCOUNT = ZCOUNT - XCOUNT
             PRINT * ,ZCOUNT,YCOUNT
            CALL HYDRMN
      C
      C --- EXIT POINT FROM PROGRAM ---------------------------------------
      C
      C     --------
            STOP 777
      C     --------
      C
      C --- FORMATS -------------------------------------------------------
      C
        101 FORMAT(1H ,'ICOND=',I2,5X,'ICONP=',I2,5X,'IOPTN=',I2,/,1X,
           .          'XMCHIN=',F13.6,5X,'RIN=',F13.6,5X,'PIN=',F13.6,/,1X,
           .          'ALFA=',F13.6,5X,'HRGG=',F13.6,5X,'IHRN=',I2,5X,/,1X,
           .          'NTIME=',I2,5X,'NDUMP=',I5,5X,'IOPORD=',I2)

             END
      C
```

```
      SUBROUTINE HYDRFL
C
C---------------------------------------------------------------I
C                                                               I
C      HYDRFL IS A 2 DIMENSIONAL RIEMANN SOLVER THAT INTEGRATES  I
C             FLUXES ACROSS NORMAL INTERFACES TO UPDATE VERTICES I
C             VARIABLES .                                       I
C                                                               I
C---------------------------------------------------------------I
C
      include     'dmsh00.h'
      include     'dhydm0.h'
      include     'dphsm0.h'
      include     'dmtrl0.h'
C
      REAL DELP(128),WSOP(128),WSOM(128),WSOO(128),
     .     RSTAR(128),CSTAR(128),PMAX(128),PMIN(128)
      REAL RRIGHT(128),URIGHT(128),VRIGHT(128),PRIGHT(128)
      REAL RLEFTT(128),ULEFTT(128),VLEFTT(128),PLEFTT(128)
      REAL ENRGYI(128),ANRGYI(128)
C
C --- BEGIN LOOP OVER ALL EDGES IN THE DOMAIN ----------------------
C
      DO 280 IH = 1 , 6
      DO 280 IC = 1 , NC
      HYDFLX( IC , IH ) = 0.
  280 CONTINUE
C
      NS1 = 1
      NS2 = NOFVES( 1 )
      DO 110 INS = 1 , NVEES
C
C --- FETCH HYDRO QUANTITIES -------------------------------------
C
      DO 120 IS = NS1 , NS2
          KS = IS - NS1 + 1
C
          RRR( KS ) = RR( IS )
          UUR( KS ) = UR( IS )
          VVR( KS ) = VR( IS )
          WWR( KS ) = WR( IS )
          PPR( KS ) = PR( IS )
          AAR( KS ) = AR( IS )
          EER( KS ) = ER( IS )
          GGR( KS ) = GR( IS )
C
          RRL( KS ) = RL( IS )
          UUL( KS ) = UL( IS )
          VVL( KS ) = VL( IS )
          WWL( KS ) = WL( IS )
          PPL( KS ) = PL( IS )
          AAL( KS ) = AL( IS )
          EEL( KS ) = EL( IS )
          GGL( KS ) = GL( IS )
C
  120 CONTINUE
C
      DO 130 KS = 1 , NOFVES( INS )
C
C --- THIS SECTION OF CODE SOLVES FOR "PSTAR" AND "USTAR" IN
C     THE RIEMANN PROBLEM USING NEWTON'S METHOD.
C
          WLEFT( KS ) = SQRT( GGL( KS ) * PPL( KS ) * RRL( KS ) )
          WRIGT( KS ) = SQRT( GGR( KS ) * PPR( KS ) * RRR( KS ) )
          WLESQ( KS ) = WLEFT( KS ) * WLEFT( KS )
          WRISQ( KS ) = WRIGT( KS ) * WRIGT( KS )
C
          PMIN( KS ) = AMIN1( PPL( KS ) , PPR( KS ) )
          PSML( KS ) = HRSM * PMIN( KS )
C
C --- FORM THE STARTING GUESS FOR THE SOLUTION ------------------
C
          PSTAR( KS ) = ( WLEFT( KS ) * PPR( KS ) +
     .                    WRIGT( KS ) * PPL( KS ) -
```

```
     .                       WLEFT( KS ) * WRIGT( KS ) *
     .                     ( UUR( KS ) - UUL( KS ) ) ) /
     .                     ( WLEFT( KS ) + WRIGT( KS ) )
          PSTAR( KS ) = AMAX1( PSTAR( KS ) , PSML( KS ) )
  130  CONTINUE
C
          DO 140 I = 1 , IHRN
C
C --- BEGIN THE NEWTON ITERATION --------------------------------
C
       DO 150 KS = 1 , NOFVES( INS )
             CFFL = ( GGL( KS ) + 1. ) / ( 2. * GGL( KS ) )
             WLEFS( KS ) = ( 1. + CFFL * ( PSTAR( KS ) /
     .                           PPL( KS ) - 1. ) ) * WLESQ( KS )
             WLEFT( KS ) = SQRT( WLEFS( KS ) )
             ZLEFT( KS ) = 2. * WLEFT( KS ) * WLEFS( KS ) /
     .                        ( WLESQ( KS ) + WLEFS( KS ) )
             USTL( KS )  = UUL( KS ) -
     .                  ( PSTAR( KS ) - PPL( KS ) ) / WLEFT( KS )
  150  CONTINUE
C
       DO 152 KS = 1 , NOFVES( INS )
             CFFR = ( GGR( KS ) + 1. ) / ( 2. * GGR( KS ) )
             WRIFS( KS ) = ( 1. + CFFR * ( PSTAR( KS ) /
     .                           PPR( KS ) - 1. ) ) * WRISQ( KS )
             WRIGT( KS ) = SQRT( WRIFS( KS ) )
             ZRIGT( KS ) = 2. * WRIGT( KS ) * WRIFS( KS ) /
     .                        ( WRISQ( KS ) + WRIFS( KS ) )
             USTR( KS )  = UUR( KS ) +
     .                  ( PSTAR( KS ) - PPR( KS ) ) / WRIGT( KS )
  152  CONTINUE
C
       DO 160 KS = 1 , NOFVES( INS )
             DPST( KS )  = ZLEFT( KS ) * ZRIGT( KS ) *
     .                  ( USTR( KS ) - USTL( KS ) ) /
     .                  ( ZLEFT( KS ) + ZRIGT( KS ) )
             PSTAR( KS ) = PSTAR( KS ) - DPST( KS )
             PSTAR( KS ) = AMAX1( PSTAR( KS ) , PSML( KS ) )
  160     CONTINUE
C
  140     CONTINUE
C
C --- FORM FINAL SOLUTIONS --------------------------------
C
       DO 170 KS = 1 , NOFVES( INS )
          CFFL = ( GGL( KS ) + 1. ) / ( 2. * GGL( KS ) )
          WLEFT( KS ) = SQRT( WLESQ( KS ) * ( 1. +
     .                  CFFL * ( PSTAR( KS ) / PPL( KS ) - 1. ) ) )
  170  CONTINUE
C
       DO 172 KS = 1 , NOFVES( INS )
          CFFR = ( GGR( KS ) + 1. ) / ( 2. * GGR( KS ) )
          WRIGT( KS ) = SQRT( WRISQ( KS ) * ( 1. +
     .                  CFFR * ( PSTAR( KS ) / PPR( KS ) - 1. ) ) )
  172  CONTINUE
C
       DO 180 KS = 1 , NOFVES( INS )
          USTAR( KS ) = ( PPL( KS ) - PPR( KS ) +
     .                     WLEFT( KS ) * UUL( KS ) +
     .                     WRIGT( KS ) * UUR( KS ) ) /
     .                  ( WLEFT( KS ) + WRIGT( KS ) )
  180  CONTINUE
C
       DO 190 KS = 1 , NOFVES( INS )
C
C --- BEGIN PROCEDURE TO OBTAIN FLUXES FROM REIMANN FORMALISM --
C
          IF( USTAR( KS ) . LE . 0.0 ) THEN
C
            RO( KS ) = RRR( KS )
            PO( KS ) = PPR( KS )
            UO( KS ) = UUR( KS )
            CO( KS ) = SQRT( HRGG * PPR( KS ) / RRR( KS ) )
            WO( KS ) = WRIGT( KS )
```

```
                GO( KS ) = GGR( KS )
                ISN( KS ) = 1
C
                ENRGYI( KS ) = EER( KS )
                ANRGYI( KS ) = AAR( KS )
                VGDNV( KS ) = VVR( KS )
                WGDNV( KS ) = WWR( KS )
C
              ELSE
C
                RO( KS ) = RRL( KS )
                PO( KS ) = PPL( KS )
                UO( KS ) = UUL( KS )
                CO( KS ) = SQRT( HRGG * PPL( KS ) / RRL( KS ) )
                WO( KS ) = WLEFT( KS )
                GO( KS ) = GGL( KS )
                ISN( KS ) = - 1
C
                ENRGYI( KS ) = EEL( KS )
                ANRGYI( KS ) = AAL( KS )
                VGDNV( KS ) = VVL( KS )
                WGDNV( KS ) = WWL( KS )
              END IF
  190   CONTINUE
C
        DO 200 KS = 1 , NOFVES( INS )
          DELP( KS ) = PSTAR( KS ) - PO( KS )
          WSOP( KS ) = ISN( KS ) * UO( KS ) + WO( KS ) / RO( KS )
          WSOM( KS ) = ISN( KS ) * UO( KS ) + CO( KS )
  200    CONTINUE
C
        DO 210 KS = 1 , NOFVES( INS )
         IF( DELP( KS ) . GT . 0. ) THEN
          WSOO( KS ) = WSOP( KS )
         ELSE
          WSOO( KS ) = WSOM( KS )
         END IF
  210   CONTINUE
C
        DO 220 KS = 1 , NOFVES( INS )
C
C --- USE OUTER STATE SOLUTION ----------------------------
C
          PGDNV( KS ) = PO( KS )
          UGDNV( KS ) = UO( KS )
          CGDNV( KS ) = CO( KS )
          RGDNV( KS ) = RO( KS )
  220   CONTINUE
C
C --- COMPUTE STARRED VALUES ---------------------------------
C
        DO 230 KS = 1 , NOFVES( INS )
          RSTAR( KS ) = 1. / ( 1. / RO( KS ) - DELP( KS ) /
     .                        ( WO( KS ) * WO( KS ) ) )
          CSTAR( KS ) = SQRT( GO( KS ) * PSTAR( KS ) / RSTAR( KS ) )
          WSOM( KS )  = ISN( KS ) * USTAR( KS ) + CSTAR( KS )
  230   CONTINUE
C
        DO 240 KS = 1 , NOFVES( INS )
         IF( DELP( KS ) . GT . 0. ) THEN
          SPIN( KS ) = WSOP( KS )
         ELSE
          SPIN( KS ) = WSOM( KS )
         END IF
  240   CONTINUE
C
        DO 250 KS = 1 , NOFVES( INS )
C
        IF( WSOO( KS ) . GE . 0. ) THEN
          IF( SPIN( KS ) . GE . 0. ) THEN
C
C --- USE THE STARRED STATE RESULTS ---------------------------
C
                      RGDNV( KS ) = RSTAR( KS )
```

```
                  UGDNV( KS ) = USTAR( KS )
                  CGDNV( KS ) = CSTAR( KS )
                  PGDNV( KS ) = PSTAR( KS )
            ELSE
C
C --- EVALUATE THE INSIDE RAREFACTION WAVE ---------------------
C
            CGDNV( KS ) = ( CSTAR( KS ) * 2. -
     .                    ISN( KS ) * USTAR( KS ) * ( GO( KS ) - 1. ) )
     .                                            / ( GO( KS ) + 1. )
            UGDNV( KS ) = - ISN( KS ) * CGDNV( KS )
            RGDNV( KS ) = ( CGDNV( KS ) / CO( KS ) ) **
     .                    ( 2. / ( GO( KS ) - 1. ) ) * RO( KS )
            PGDNV( KS ) = CGDNV( KS ) * CGDNV( KS ) * RGDNV( KS ) /
     .                                                   GO( KS )
C
            END IF
C
         END IF
  250 CONTINUE
C
      DO 260 KS = 1 , NOFVES( INS )
         IS = KS + NS1 - 1
C
      ICL = JS( 7 , IS )
      ICR = JS( 8 , IS )
C
      CTT = SQRT( GO( KS ) * PGDNV( KS ) / RGDNV( KS ) )
      XSS = XS( 5 , IS )
      XYZ = 1. / XSS
C
      IATRB = JS( 9 , IS )
      IF( IATRB . EQ . 0 ) THEN
C
         XXN = ( XC( 1 , ICR ) - XC( 1 , ICL ) ) * XYZ
         YYN = ( XC( 2 , ICR ) - XC( 2 , ICL ) ) * XYZ
         ZZN = ( XC( 3 , ICR ) - XC( 3 , ICL ) ) * XYZ
C
      VEL =
     .      ( UGDNV( KS ) * XN( IS ) +
     .        VGDNV( KS ) * XP( IS ) +
     .        WGDNV( KS ) * XT( IS ) ) * XXN +
     .      ( UGDNV( KS ) * YN( IS ) +
     .        VGDNV( KS ) * YP( IS ) +
     .        WGDNV( KS ) * YT( IS ) ) * YYN +
     .      ( UGDNV( KS ) * ZN( IS ) +
     .        VGDNV( KS ) * ZP( IS ) +
     .        WGDNV( KS ) * ZT( IS ) ) * ZZN
C
      DTU = XSS / ( CTT + ABS( VEL ) )
      DTT = AMIN1( DTU , DTT )
C
      ELSE
C
         XXN = ( XYZMDL( 1 , IS ) - XC( 1 , ICL ) ) * XYZ
         YYN = ( XYZMDL( 2 , IS ) - XC( 2 , ICL ) ) * XYZ
         ZZN = ( XYZMDL( 3 , IS ) - XC( 3 , ICL ) ) * XYZ
C
      VEL =
     .      ( UGDNV( KS ) * XN( IS ) +
     .        VGDNV( KS ) * XP( IS ) +
     .        WGDNV( KS ) * XT( IS ) ) * XXN +
     .      ( UGDNV( KS ) * YN( IS ) +
     .        VGDNV( KS ) * YP( IS ) +
     .        WGDNV( KS ) * YT( IS ) ) * YYN +
     .      ( UGDNV( KS ) * ZN( IS ) +
     .        VGDNV( KS ) * ZP( IS ) +
     .        WGDNV( KS ) * ZT( IS ) ) * ZZN
C
      DTU = XSS / ( CTT + ABS( VEL ) )
      DTT = AMIN1( DTU , DTT )
C
      END IF
  260 CONTINUE
```

```
C
      DO 270 KS = 1 , NOFVES( INS )
         IS = KS + NS1 - 1
C
C ... FLUX FOR DENSITY .............................................
C
      RO( KS ) = RGDNV( KS ) * UGDNV( KS )
C
C ... FLUX FOR MOMENTUM DENSITY ....................................
C
      UO( KS ) = PGDNV( KS ) * XN( IS ) +
     .           RO( KS ) * ( UGDNV( KS ) * XN( IS ) +
     .                        VGDNV( KS ) * XP( IS ) +
     .                        WGDNV( KS ) * XT( IS ) )
      CO( KS ) = PGDNV( KS ) * YN( IS ) +
     .           RO( KS ) * ( UGDNV( KS ) * YN( IS ) +
     .                        VGDNV( KS ) * YP( IS ) +
     .                        WGDNV( KS ) * YT( IS ) )
      WO( KS ) = PGDNV( KS ) * ZN( IS ) +
     .           RO( KS ) * ( UGDNV( KS ) * ZN( IS ) +
     .                        VGDNV( KS ) * ZP( IS ) +
     .                        WGDNV( KS ) * ZT( IS ) )
C
C ... FLUX FOR ENERGY DENSITY ......................................
C
      PO( KS ) = UGDNV( KS ) * ( ENRGYI( KS ) +
     .      .5 * RGDNV( KS ) * ( UGDNV( KS ) * UGDNV( KS ) +
     .                           VGDNV( KS ) * VGDNV( KS ) +
     .                           WGDNV( KS ) * WGDNV( KS ) ) )
C
C ... FLUX FOR COMBUSTION INTERFACE TRACKING .......................
C
      AO( KS ) = UGDNV( KS ) * RGDNV( KS ) * ANRGYI( KS )
C
  270 CONTINUE
C
      DO 290 IS = NS1 , NS2
         KS = IS - NS1 + 1
C
       ICL = JS( 7 , IS )
       ICR = JS( 8 , IS )
C
      IATRB = JS( 9 , IS )
      IF( IATRB . EQ . 0 ) THEN
C
C ... FLUX FOR DENSITY .............................................
C
      DLENG = XS( 4 , IS ) * RO( KS )
      HYDFLX( ICL , 1 ) = HYDFLX( ICL , 1 ) + DLENG
      HYDFLX( ICR , 1 ) = HYDFLX( ICR , 1 ) - DLENG
C
C ... FLUX FOR MOMENTUM DENSITY ( U DIRECTION ) ....................
C
      DLENG = XS( 4 , IS ) * UO( KS )
      HYDFLX( ICL , 2 ) = HYDFLX( ICL , 2 ) + DLENG
      HYDFLX( ICR , 2 ) = HYDFLX( ICR , 2 ) - DLENG
C
C ... FLUX FOR MOMENTUM DENSITY ( V DIRECTION ) ....................
C
      DLENG = XS( 4 , IS ) * CO( KS )
      HYDFLX( ICL , 3 ) = HYDFLX( ICL , 3 ) + DLENG
      HYDFLX( ICR , 3 ) = HYDFLX( ICR , 3 ) - DLENG
C
C ... FLUX FOR MOMENTUM DENSITY ( W DIRECTION ) ....................
C
      DLENG = XS( 4 , IS ) * WO( KS )
      HYDFLX( ICL , 4 ) = HYDFLX( ICL , 4 ) + DLENG
      HYDFLX( ICR , 4 ) = HYDFLX( ICR , 4 ) - DLENG
C
C ... FLUX FOR ENERGY DENSITY ......................................
C
      DLENG = XS( 4 , IS ) * PO( KS )
      HYDFLX( ICL , 5 ) = HYDFLX( ICL , 5 ) + DLENG
      HYDFLX( ICR , 5 ) = HYDFLX( ICR , 5 ) - DLENG
```

```
C
C ... FLUX FOR COMBUSTION INTERFACE TRACKING.........................
C
      DLENG = XS( 4 , IS ) * AO( KS )
      HYDFLX( ICL , 6 ) = HYDFLX( ICL , 6 ) + DLENG
      HYDFLX( ICR , 6 ) = HYDFLX( ICR , 6 ) - DLENG
C
      ELSE
C
C ... FLUX FOR DENSITY ...............................................
C
      DLENG = XS( 4 , IS ) * RO( KS )
      HYDFLX( ICL , 1 ) = HYDFLX( ICL , 1 ) + DLENG
C
C ... FLUX FOR MOMENTUM DENSITY ( U DIRECTION ) ......................
C
      DLENG = XS( 4 , IS ) * UO( KS )
      HYDFLX( ICL , 2 ) = HYDFLX( ICL , 2 ) + DLENG
C
C ... FLUX FOR MOMENTUM DENSITY ( V DIRECTION ) ......................
C
      DLENG = XS( 4 , IS ) * CO( KS )
      HYDFLX( ICL , 3 ) = HYDFLX( ICL , 3 ) + DLENG
C
C ... FLUX FOR MOMENTUM DENSITY ( W DIRECTION ) ......................
C
      DLENG = XS( 4 , IS ) * WO( KS )
      HYDFLX( ICL , 4 ) = HYDFLX( ICL , 4 ) + DLENG
C
C ... FLUX FOR ENERGY DENSITY ........................................
C
      DLENG = XS( 4 , IS ) * PO( KS )
      HYDFLX( ICL , 5 ) = HYDFLX( ICL , 5 ) + DLENG
C
C ... FLUX FOR COMBUSTION INTERFACE TRACKING.........................
C
      DLENG = XS( 4 , IS ) * AO( KS )
      HYDFLX( ICL , 6 ) = HYDFLX( ICL , 6 ) + DLENG
C
      END IF
  290 CONTINUE
C
      NS1 = NS2 + 1
      NS2 = NS2 + NOFVES( INS + 1 )
  110 CONTINUE
C
      RETURN
      END
C
```

```
      SUBROUTINE RYDRFL
C
C-------------------------------------------------------------I
C                                                             I
C     RYDRFL IS A 2 DIMENSIONAL RIEMANN SOLVER THAT INTEGRATES I
C            FLUXES ACROSS NORMAL INTERFACES TO UPDATE VERTICES I
C            VARIABLES .                                      I
C                                                             I
C-------------------------------------------------------------I
C
      include    'dmsh00.h'
      include    'dhydm0.h'
      include    'dphsm0.h'
      include    'dmtrl0.h'
C
      REAL DELP(128),WSOP(128),WSOM(128),WSOO(128),
     .     RSTAR(128),CSTAR(128),PMAX(128),PMIN(128)
      REAL RRIGHT(128),URIGHT(128),VRIGHT(128),PRIGHT(128)
      REAL RLEFTT(128),ULEFTT(128),VLEFTT(128),PLEFTT(128)
      REAL ENRGYI(128),ANRGYI(128)
C
C --- BEGIN LOOP OVER ALL EDGES IN THE DOMAIN ----------------------
C
      NS1 = 1
      NS2 = NOFVES( 1 )
      DO 110 INS = 1 , NVEES
C
C --- FETCH HYDRO QUANTITIES -------------------------------------
C
      DO 120 IS = NS1 , NS2
          KS = IS - NS1 + 1
C
          ICL = JS( 7 , IS )
          IBC = JS( 9 , IS )
C
          RRL( KS ) = HYDV( ICL , 1 )
          UUL( KS ) = HYDV( ICL , 2 ) * XN( IS ) +
     .                HYDV( ICL , 3 ) * YN( IS ) +
     .                HYDV( ICL , 4 ) * ZN( IS )
          VVL( KS ) = HYDV( ICL , 2 ) * XP( IS ) +
     .                HYDV( ICL , 3 ) * YP( IS ) +
     .                HYDV( ICL , 4 ) * ZP( IS )
          WWL( KS ) = HYDV( ICL , 2 ) * XT( IS ) +
     .                HYDV( ICL , 3 ) * YT( IS ) +
     .                HYDV( ICL , 4 ) * ZT( IS )
          PPL( KS ) = HYDV( ICL , 5 )
          AAL( KS ) = HYDV( ICL , 6 )
          EEL( KS ) = HYDV( ICL , 8 )
          GGL( KS ) = HYDV( ICL , 7 )
C
          RRR( KS ) = RRL( KS )
          IF( IBC . EQ . 0 ) THEN
          UUR( KS ) = UUL( KS )
          ELSE
          UUR( KS ) = - UUL( KS )
          END IF
          VVR( KS ) = VVL( KS )
          WWR( KS ) = WWL( KS )
          PPR( KS ) = PPL( KS )
          AAR( KS ) = AAL( KS )
          EER( KS ) = EEL( KS )
          GGR( KS ) = GGL( KS )
C
  120 CONTINUE
C
      DO 130 KS = 1 , NOFVES( INS )
C
C --- THIS SECTION OF CODE SOLVES FOR "PSTAR" AND "USTAR" IN
C     THE RIEMANN PROBLEM USING NEWTON'S METHOD.
C
          WLEFT( KS ) = SQRT( GGL( KS ) * PPL( KS ) * RRL( KS ) )
          WRIGT( KS ) = SQRT( GGR( KS ) * PPR( KS ) * RRR( KS ) )
          WLESQ( KS ) = WLEFT( KS ) * WLEFT( KS )
          WRISQ( KS ) = WRIGT( KS ) * WRIGT( KS )
```

```fortran
C
            PMIN( KS ) = AMIN1( PPL( KS ) , PPR( KS ) )
            PSML( KS ) = HRSM * PMIN( KS )
C
C --- FORM THE STARTING GUESS FOR THE SOLUTION -------------------
C
            PSTAR( KS ) = ( WLEFT( KS ) * PPR( KS ) +
     .                       WRIGT( KS ) * PPL( KS ) -
     .                       WLEFT( KS ) * WRIGT( KS ) *
     .                     ( UUR( KS ) - UUL( KS ) ) ) /
     .                     ( WLEFT( KS ) + WRIGT( KS ) )
            PSTAR( KS ) = AMAX1( PSTAR( KS ) , PSML( KS ) )
  130  CONTINUE
C
            DO 140 I = 1 , IHRN
C
C --- BEGIN THE NEWTON ITERATION --------------------------------
C
       DO 150 KS = 1 , NOFVES( INS )
              CFFL = ( GGL( KS ) + 1. ) / ( 2. * GGL( KS ) )
              WLEFS( KS ) = ( 1. + CFFL * ( PSTAR( KS ) /
     .                          PPL( KS ) - 1. ) ) * WLESQ( KS )
              WLEFT( KS ) = SQRT( WLEFS( KS ) )
              ZLEFT( KS ) = 2. * WLEFT( KS ) * WLEFS( KS ) /
     .                     ( WLESQ( KS ) + WLEFS( KS ) )
              USTL( KS )  = UUL( KS ) -
     .                 ( PSTAR( KS ) - PPL( KS ) ) / WLEFT( KS )
  150  CONTINUE
C
       DO 152 KS = 1 , NOFVES( INS )
              CFFR = ( GGR( KS ) + 1. ) / ( 2. * GGR( KS ) )
              WRIFS( KS ) = ( 1. + CFFR * ( PSTAR( KS ) /
     .                          PPR( KS ) - 1. ) ) * WRISQ( KS )
              WRIGT( KS ) = SQRT( WRIFS( KS ) )
              ZRIGT( KS ) = 2. * WRIGT( KS ) * WRIFS( KS ) /
     .                     ( WRISQ( KS ) + WRIFS( KS ) )
              USTR( KS )  = UUR( KS ) +
     .                 ( PSTAR( KS ) - PPR( KS ) ) / WRIGT( KS )
  152  CONTINUE
C
       DO 160 KS = 1 , NOFVES( INS )
              DPST( KS )  = ZLEFT( KS ) * ZRIGT( KS ) *
     .                 ( USTR( KS ) - USTL( KS ) ) /
     .                 ( ZLEFT( KS ) + ZRIGT( KS ) )
              PSTAR( KS ) = PSTAR( KS ) - DPST( KS )
              PSTAR( KS ) = AMAX1( PSTAR( KS ) , PSML( KS ) )
  160     CONTINUE
C
  140     CONTINUE
C
C --- FORM FINAL SOLUTIONS ----------------------------------
C
       DO 170 KS = 1 , NOFVES( INS )
           CFFL = ( GGL( KS ) + 1. ) / ( 2. * GGL( KS ) )
           WLEFT( KS ) = SQRT( WLESQ( KS ) * ( 1. +
     .                 CFFL * ( PSTAR( KS ) / PPL( KS ) - 1. ) ) )
  170  CONTINUE
C
       DO 172 KS = 1 , NOFVES( INS )
           CFFR = ( GGR( KS ) + 1. ) / ( 2. * GGR( KS ) )
           WRIGT( KS ) = SQRT( WRISQ( KS ) * ( 1. +
     .                 CFFR * ( PSTAR( KS ) / PPR( KS ) - 1. ) ) )
  172  CONTINUE
C
       DO 180 KS = 1 , NOFVES( INS )
           USTAR( KS ) = ( PPL( KS ) - PPR( KS ) +
     .                    WLEFT( KS ) * UUL( KS ) +
     .                    WRIGT( KS ) * UUR( KS ) ) /
     .                 ( WLEFT( KS ) + WRIGT( KS ) )
  180  CONTINUE
C
       DO 190 KS = 1 , NOFVES( INS )
C
C --- BEGIN PROCEDURE TO OBTAIN FLUXES FROM REIMANN FORMALISM --
```

```
C
            IF( USTAR( KS ) . LE . 0.0 ) THEN
C
              RO( KS ) = RRR( KS )
              PO( KS ) = PPR( KS )
              UO( KS ) = UUR( KS )
              CO( KS ) = SQRT( HRGG * PPR( KS ) / RRR( KS ) )
              WO( KS ) = WRIGT( KS )
              GO( KS ) = GGR( KS )
              ISN( KS ) = 1
C
              ENRGYI( KS ) = EER( KS )
              ANRGYI( KS ) = AAR( KS )
              VGDNV( KS ) = VVR( KS )
              WGDNV( KS ) = WWR( KS )
C
            ELSE
C
              RO( KS ) = RRL( KS )
              PO( KS ) = PPL( KS )
              UO( KS ) = UUL( KS )
              CO( KS ) = SQRT( HRGG * PPL( KS ) / RRL( KS ) )
              WO( KS ) = WLEFT( KS )
              GO( KS ) = GGL( KS )
              ISN( KS ) = - 1
C
              ENRGYI( KS ) = EEL( KS )
              ANRGYI( KS ) = AAL( KS )
              VGDNV( KS ) = VVL( KS )
              WGDNV( KS ) = WWL( KS )
            END IF
  190 CONTINUE
C
      DO 200 KS = 1 , NOFVES( INS )
        DELP( KS ) = PSTAR( KS ) - PO( KS )
        WSOP( KS ) = ISN( KS ) * UO( KS ) + WO( KS ) / RO( KS )
        WSOM( KS ) = ISN( KS ) * UO( KS ) + CO( KS )
  200  CONTINUE
C
      DO 210 KS = 1 , NOFVES( INS )
        IF( DELP( KS ) . GT . 0. ) THEN
          WSOO( KS ) = WSOP( KS )
        ELSE
          WSOO( KS ) = WSOM( KS )
        END IF
  210 CONTINUE
C
      DO 220 KS = 1 , NOFVES( INS )
C
C --- USE OUTER STATE SOLUTION ----------------------------
C
          PGDNV( KS ) = PO( KS )
          UGDNV( KS ) = UO( KS )
          CGDNV( KS ) = CO( KS )
          RGDNV( KS ) = RO( KS )
  220 CONTINUE
C
C --- COMPUTE STARRED VALUES -------------------------------
C
      DO 230 KS = 1 , NOFVES( INS )
          RSTAR( KS ) = 1. / ( 1. / RO( KS ) - DELP( KS ) /
     .                       ( WO( KS ) * WO( KS ) ) )
          CSTAR( KS ) = SQRT( GO( KS ) * PSTAR( KS ) / RSTAR( KS ) )
          WSOM( KS )  = ISN( KS ) * USTAR( KS ) + CSTAR( KS )
  230 CONTINUE
C
      DO 240 KS = 1 , NOFVES( INS )
        IF( DELP( KS ) . GT . 0. ) THEN
          SPIN( KS ) = WSOP( KS )
        ELSE
          SPIN( KS ) = WSOM( KS )
        END IF
  240 CONTINUE
C
```

```
      DO 250 KS = 1 , NOFVES( INS )
C
      IF( WS00( KS ) . GE . 0. ) THEN
         IF( SPIN( KS ) . GE . 0. ) THEN
C
C --- USE THE STARRED STATE RESULTS ---------------------------
C
                        RGDNV( KS ) = RSTAR( KS )
                        UGDNV( KS ) = USTAR( KS )
                        CGDNV( KS ) = CSTAR( KS )
                        PGDNV( KS ) = PSTAR( KS )
         ELSE
C
C --- EVALUATE THE INSIDE RAREFACTION WAVE --------------------
C
         CGDNV( KS ) = ( CSTAR( KS ) * 2. -
     .                  ISN( KS ) * USTAR( KS ) * ( GO( KS ) - 1. ) )
     .                                      / ( GO( KS ) + 1. )
         UGDNV( KS ) = - ISN( KS ) * CGDNV( KS )
         RGDNV( KS ) = ( CGDNV( KS ) / CO( KS ) ) **
     .                ( 2. / ( GO( KS ) - 1. ) ) * RO( KS )
         PGDNV( KS ) = CGDNV( KS ) * CGDNV( KS ) * RGDNV( KS ) /
     .                                            GO( KS )
C
         END IF
C
      END IF
  250 CONTINUE
C
      DO 260 KS = 1 , NOFVES( INS )
         IS = KS + NS1 - 1
         RR( IS ) = RGDNV( KS )
         PR( IS ) = PGDNV( KS )
  260 CONTINUE
C
      NS1 = NS2 + 1
      NS2 = NS2 + NOFVES( INS + 1 )
  110 CONTINUE
C
      RETURN
      END
C
```

Thu Jul 1 14:17:00 1993   threed.f   SUBROUTINE KYDRFL

```
      SUBROUTINE KYDRFL
C
C-------------------------------------------------------------I
C                                                             I
C      KYDRFL IS A 2 DIMENSIONAL RIEMANN SOLVER THAT INTEGRATES I
C             FLUXES ACROSS NORMAL INTERFACES TO UPDATE VERTICES I
C             VARIABLES .                                     I
C                                                             I
C-------------------------------------------------------------I
C
      include     'dmsh00.h'
      include     'dhydm0.h'
      include     'dphsm0.h'
      include     'dmtrl0.h'
C
      REAL DELP(128),WSOP(128),WSOM(128),WS00(128),
     .     RSTAR(128),CSTAR(128),PMAX(128),PMIN(128)
      REAL RRIGHT(128),URIGHT(128),VRIGHT(128),PRIGHT(128)
      REAL RLEFTT(128),ULEFTT(128),VLEFTT(128),PLEFTT(128)
      INTEGER NOFVEP(128)
C
C --- FETCH HYDRO QUANTITIES -------------------------------------
C
      DO 120 KS = 1 , NPRTCL
         IS = IJKPRT( KS )
         ICL = JS( 7 , IS )
         IBC = JS( 9 , IS )
C
         RRL( KS ) = HYDV( ICL , 1 )
```

```
          UUL( KS ) = HYDV( ICL , 2 ) * XN( IS ) +
     .                HYDV( ICL , 3 ) * YN( IS ) +
     .                HYDV( ICL , 4 ) * ZN( IS )
          VVL( KS ) = HYDV( ICL , 2 ) * XP( IS ) +
     .                HYDV( ICL , 3 ) * YP( IS ) +
     .                HYDV( ICL , 4 ) * ZP( IS )
          WWL( KS ) = HYDV( ICL , 2 ) * XT( IS ) +
     .                HYDV( ICL , 3 ) * YT( IS ) +
     .                HYDV( ICL , 4 ) * ZT( IS )
          PPL( KS ) = HYDV( ICL , 5 )
          AAL( KS ) = HYDV( ICL , 6 )
          EEL( KS ) = HYDV( ICL , 8 )
          GGL( KS ) = HYDV( ICL , 7 )
C
          RRR( KS ) = RRL( KS )
          IF( IBC . EQ . 0 ) THEN
          UUR( KS ) = UUL( KS )
          ELSE
          UUR( KS ) = - UUL( KS )
          END IF
          VVR( KS ) = VVL( KS )
          WWR( KS ) = WWL( KS )
          PPR( KS ) = PPL( KS )
          AAR( KS ) = AAL( KS )
          EER( KS ) = EEL( KS )
          GGR( KS ) = GGL( KS )
C
  120 CONTINUE
C
      DO 130 KS = 1 , NPRTCL
C
C --- THIS SECTION OF CODE SOLVES FOR "PSTAR" AND "USTAR" IN
C     THE RIEMANN PROBLEM USING NEWTON'S METHOD.
C
          WLEFT( KS ) = SQRT( GGL( KS ) * PPL( KS ) * RRL( KS ) )
          WRIGT( KS ) = SQRT( GGR( KS ) * PPR( KS ) * RRR( KS ) )
          WLESQ( KS ) = WLEFT( KS ) * WLEFT( KS )
          WRISQ( KS ) = WRIGT( KS ) * WRIGT( KS )
C
          PMIN( KS ) = AMIN1( PPL( KS ) , PPR( KS ) )
          PSML( KS ) = HRSM * PMIN( KS )
C
C --- FORM THE STARTING GUESS FOR THE SOLUTION ------------------
C
          PSTAR( KS ) = ( WLEFT( KS ) * PPR( KS ) +
     .                    WRIGT( KS ) * PPL( KS ) -
     .                    WLEFT( KS ) * WRIGT( KS ) *
     .                  ( UUR( KS ) - UUL( KS ) ) ) /
     .                  ( WLEFT( KS ) + WRIGT( KS ) )
          PSTAR( KS ) = AMAX1( PSTAR( KS ) , PSML( KS ) )
  130 CONTINUE
C
          DO 140 I = 1 , IHRN
C
C --- BEGIN THE NEWTON ITERATION -------------------------------
C
      DO 150 KS = 1 , NPRTCL
            CFFL = ( GGL( KS ) + 1. ) / ( 2. * GGL( KS ) )
            WLEFS( KS ) = ( 1. + CFFL * ( PSTAR( KS ) /
     .                      PPL( KS ) - 1. ) ) * WLESQ( KS )
            WLEFT( KS ) = SQRT( WLEFS( KS ) )
            ZLEFT( KS ) = 2. * WLEFT( KS ) * WLEFS( KS ) /
     .                  ( WLESQ( KS ) + WLEFS( KS ) )
            USTL( KS )  = UUL( KS ) -
     .                ( PSTAR( KS ) - PPL( KS ) ) / WLEFT( KS )
  150 CONTINUE
C
      DO 152 KS = 1 , NPRTCL
            CFFR = ( GGR( KS ) + 1. ) / ( 2. * GGR( KS ) )
            WRIFS( KS ) = ( 1. + CFFR * ( PSTAR( KS ) /
     .                      PPR( KS ) - 1. ) ) * WRISQ( KS )
            WRIGT( KS ) = SQRT( WRIFS( KS ) )
            ZRIGT( KS ) = 2. * WRIGT( KS ) * WRIFS( KS ) /
     .                  ( WRISQ( KS ) + WRIFS( KS ) )
```

```
               USTR( KS )  = UUR( KS ) +
     .                       ( PSTAR( KS ) - PPR( KS ) ) / WRIGT( KS )
  152  CONTINUE
C
         DO 160 KS = 1 , NPRTCL
               DPST( KS )  = ZLEFT( KS ) * ZRIGT( KS ) *
     .                       ( USTR( KS ) - USTL( KS ) ) /
     .                       ( ZLEFT( KS ) + ZRIGT( KS ) )
               PSTAR( KS ) = PSTAR( KS ) - DPST( KS )
               PSTAR( KS ) = AMAX1( PSTAR( KS ) , PSML( KS ) )
  160     CONTINUE
C
  140     CONTINUE
C
C --- FORM FINAL SOLUTIONS ---------------------------------
C
         DO 170 KS = 1 , NPRTCL
            CFFL = ( GGL( KS ) + 1. ) / ( 2. * GGL( KS ) )
            WLEFT( KS ) = SQRT( WLESQ( KS ) * ( 1. +
     .                    CFFL * ( PSTAR( KS ) / PPL( KS ) - 1. ) ) )
  170  CONTINUE
C
         DO 172 KS = 1 , NPRTCL
            CFFR = ( GGR( KS ) + 1. ) / ( 2. * GGR( KS ) )
            WRIGT( KS ) = SQRT( WRISQ( KS ) * ( 1. +
     .                    CFFR * ( PSTAR( KS ) / PPR( KS ) - 1. ) ) )
  172  CONTINUE
C
         DO 180 KS = 1 , NPRTCL
            USTAR( KS ) = ( PPL( KS ) - PPR( KS ) +
     .                      WLEFT( KS ) * UUL( KS ) +
     .                      WRIGT( KS ) * UUR( KS ) ) /
     .                    ( WLEFT( KS ) + WRIGT( KS ) )
  180  CONTINUE
C
         DO 190 KS = 1 , NPRTCL
C
C --- BEGIN PROCEDURE TO OBTAIN FLUXES FROM REIMANN FORMALISM --
C
            IF( USTAR( KS ) . LE . 0.0 ) THEN
C
              RO( KS ) = RRR( KS )
              PO( KS ) = PPR( KS )
              UO( KS ) = UUR( KS )
              CO( KS ) = SQRT( HRGG * PPR( KS ) / RRR( KS ) )
              WO( KS ) = WRIGT( KS )
              GO( KS ) = GGR( KS )
              ISN( KS ) = 1
C
              VGDNV( KS ) = VVR( KS )
              WGDNV( KS ) = WWR( KS )
C
            ELSE
C
              RO( KS ) = RRL( KS )
              PO( KS ) = PPL( KS )
              UO( KS ) = UUL( KS )
              CO( KS ) = SQRT( HRGG * PPL( KS ) / RRL( KS ) )
              WO( KS ) = WLEFT( KS )
              GO( KS ) = GGL( KS )
              ISN( KS ) = - 1
C
              VGDNV( KS ) = VVL( KS )
              WGDNV( KS ) = WWL( KS )
            END IF
  190  CONTINUE
C
         DO 200 KS = 1 , NPRTCL
            DELP( KS ) = PSTAR( KS ) - PO( KS )
            WSOP( KS ) = ISN( KS ) * UO( KS ) + WO( KS ) / RO( KS )
            WSOM( KS ) = ISN( KS ) * UO( KS ) + CO( KS )
  200   CONTINUE
C
         DO 210 KS = 1 , NPRTCL
```

```
            IF( DELP( KS ) . GT . 0. ) THEN
              WSOO( KS ) = WSOP( KS )
            ELSE
              WSOO( KS ) = WSOM( KS )
            END IF
  210     CONTINUE
C
          DO 220 KS = 1 , NPRTCL
C
C --- USE OUTER STATE SOLUTION ----------------------------
C
              PGDNV( KS ) = PO( KS )
              UGDNV( KS ) = UO( KS )
              CGDNV( KS ) = CO( KS )
              RGDNV( KS ) = RO( KS )
  220     CONTINUE
C
C --- COMPUTE STARRED VALUES --------------------------------
C
          DO 230 KS = 1 , NPRTCL
              RSTAR( KS ) = 1. / ( 1. / RO( KS ) - DELP( KS ) /
     .                         ( WO( KS ) * WO( KS ) ) )
              CSTAR( KS ) = SQRT( GO( KS ) * PSTAR( KS ) / RSTAR( KS ) )
              WSOM( KS )  = ISN( KS ) * USTAR( KS ) + CSTAR( KS )
  230     CONTINUE
C
          DO 240 KS = 1 , NPRTCL
            IF( DELP( KS ) . GT . 0. ) THEN
              SPIN( KS ) = WSOP( KS )
            ELSE
              SPIN( KS ) = WSOM( KS )
            END IF
  240     CONTINUE
C
          DO 250 KS = 1 , NPRTCL
C
          IF( WSOO( KS ) . GE . 0. ) THEN
              IF( SPIN( KS ) . GE . 0. ) THEN
C
C --- USE THE STARRED STATE RESULTS --------------------------
C
                              RGDNV( KS ) = RSTAR( KS )
                              UGDNV( KS ) = USTAR( KS )
                              CGDNV( KS ) = CSTAR( KS )
                              PGDNV( KS ) = PSTAR( KS )
              ELSE
C
C --- EVALUATE THE INSIDE RAREFACTION WAVE --------------------
C
              CGDNV( KS ) = ( CSTAR( KS ) * 2. -
     .                        ISN( KS ) * USTAR( KS ) * ( GO( KS ) - 1. ) )
     .                                       / ( GO( KS ) + 1. )
              UGDNV( KS ) = - ISN( KS ) * CGDNV( KS )
              RGDNV( KS ) = ( CGDNV( KS ) / CO( KS ) ) **
     .                      ( 2. / ( GO( KS ) - 1. ) ) * RO( KS )
              PGDNV( KS ) = CGDNV( KS ) * CGDNV( KS ) * RGDNV( KS ) /
     .                                                  GO( KS )
C
              END IF
C
          END IF
  250     CONTINUE
C
          DO 260 KS = 1 , NPRTCL
              RR( KS ) = RGDNV( KS )
              PR( KS ) = PGDNV( KS )
  260     CONTINUE
C
          RETURN
          END
C
```

```
      SUBROUTINE HYDRMN
C
C-------------------------------------------------------------------I
C                                                                   I
C      HYDRMN IS A 2 DIMENSIONAL RIEMANN SOLVER THAT CALCULATES     I
C             FLUXES ACROSS NORMAL INTERFACES.                      I
C             IT IS CONFIGURED TO WORK IN EITHER TWO OR THREE       I
C             DIMENSIONAL SITUATIONS. THE HYDRODYNAMIC QUANTITIES   I
C             CAN BE SIDE OR VERTEX CENTERED FOR 2-D AND CELL OR    I
C             VERTEX CENTERED FOR 3-D. THE SPECIFIC USE IS BASED    I
C             ON THE CONTENTS OF "OPTHYD".                          I
C                                                                   I
C                                                                   I
C-------------------------------------------------------------------I
C
C      THE USE OF THE HYDRO VARIABLES IS AS FOLLOWS:
C
C      +++++++++++++++++++++++++++++++++++++++++++++++++++++++
C      +                                                     +
C      +      HYDV(IV,IH) CONTAINS VERTEX CENTERED HYDRO-    +
C      +      DYNAMIC QUANTITIES. IT IS USED WITH ALL CASES. +
C      +                                                     +
C      +      HYDE(IH,IE) CONTAINS EDGE   CENTERED HYDRO-    +
C      +      DYNAMIC FLUX QUANTITIES WITH ORIENTATION       +
C      +      DETERMINED BY THE "SIDE"  "VERTEX" OR "CELL  " +
C      +      OPTIONS . IT IS USED FOR THE CASES WHERE       +
C      +      "OPTHYD" = "SIDE  2D" 2-D SIDE CENTERED        +
C      +               = "VERTEX2D" 2-D VERTEX CENTERED      +
C      +               = "VERTEX3D" 3-D VERTEX CENTERED      +
C      +                                                     +
C      +      HYDC(IC,IH) CONTAINS CELL   CENTERED HYDRO-    +
C      +      DYNAMIC QUANTITIES. IT IS USED FOR THE CASE    +
C      +      "OPTHYD" = "CELL  3D" 3-D CELL   CENTERED      +
C      +                                                     +
C      +                                                     +
C      +         IV   - VERTEX INDEX                         +
C      +         IS   - SIDE   INDEX                         +
C      +         IE   - EDGE   INDEX                         +
C      +         IC   - CELL   INDEX                         +
C      +         IH   - HYDRO  INDEX                         +
C      +                                                     +
C      +                1 = RO  DENSITY IN ***********       +
C      +                2 = UX  X VELOCITY ***********       +
C      +                3 = UY  Y VELOCITY ***********       +
C      +                4 = UZ  ***********************       +
C      +                5 = PO  PRESSURE IN **********       +
C      +                6 = EN  ENERGY IN   **********       +
C      +                                                     +
C      +++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
      include     'dmsh00.h'
      include     'dhydm0.h'
      include     'dphsm0.h'
      include     'dmtrl0.h'
C
      REAL RRN(128),URN(128),VRN(128),WRN(128),EPN(128),
     .     ARN(128),XS2S(128),XSAR(128)
      REAL HYDVR(128),HYDVU(128),HYDVV(128),HYDVW(128),
     .     HYDVP(128)
      INTEGER NDUMMY1,NDUMMY2,NDUMMY3
      INTEGER IDUMMY(4),VDATA(2),FDATA(2)
      CHARACTER*31 VLABEL
      CHARACTER*32 FLABEL
      CHARACTER*6 CTRI, CTET
      INTEGER ISURF(400000)
C
      REAL CD1(4),CD2(4)
C
      NDUMMY1=1
      NDUMMY2=4
      NDUMMY3=0
      IDUMMY(1) = 0
      IDUMMY(2) = 0
      IDUMMY(3) = 0
```

```
      IDUMMY(4) = 0
      VDATA(1) = 1
      VDATA(2) = 1
      FDATA(1) = 1
      FDATA(2) = 4
      VLABEL=' pressure, new / m**2'
      FLABEL=' tets faces,'
      CTRI='  tri '
      CTET='  tet '
C      TLIMIT=TT
      TLIMIT=30.
C
       IJKNUM = 0
       IF( ICONP . EQ . 1 ) THEN
       REWIND 10
       REWIND 26
       READ (26,*) IJKNUM
       DO KKJ = 1 , IJKNUM
       READ (26,*) RO,(RRN(IK),IK=1,NPRTCL)
       WRITE (10,*) RO,(RRN(IK),IK=1,NPRTCL)
       END DO
       END IF
      DO 120 JT = 1 , NTIME
      IF(JT.GT.5) IEOS=0
C
        DO KK = 1 , 5
        DO IV = 1 , NV
        HNUM( IV , KK ) = 0.
        END DO
        END DO
      DO 140 ITT = 1 , NDUMP
C
C --- SELECT ORDER OF INTEGRATION ------------------------------------
C
      IF(IOPORD.EQ.1)THEN
         CALL FIRST
      ELSEIF(IOPORD.EQ.2)THEN
         CALL GRADNT
      ENDIF
C
      DTT = 1.E24
C
      CALL HYDRFL
C
      DTT = DTT * .4
      TT = TT + DTT
      PRINT *,JT,ITT,DTT,TT,NS
C
      NC1 = 1
      NC2 = NOFVEC( 1 )
      DO 110 INC = 1 , NVEEC
C
      DO 150 IC = NC1 , NC2
         KC = IC - NC1 + 1
         RRR( KC ) = HYDV( IC , 1 )
         UUR( KC ) = HYDV( IC , 2 )
         VVR( KC ) = HYDV( IC , 3 )
         WWR( KC ) = HYDV( IC , 4 )
         PPR( KC ) = HYDV( IC , 5 )
         AAR( KC ) = HYDV( IC , 6 )
C
         RRL( KC ) = HYDFLX( IC , 1 )
         UUL( KC ) = HYDFLX( IC , 2 )
         VVL( KC ) = HYDFLX( IC , 3 )
         WWL( KC ) = HYDFLX( IC , 4 )
         PPL( KC ) = HYDFLX( IC , 5 )
         AAL( KC ) = HYDFLX( IC , 6 )
C
         XS2S( KC ) = XC( 2 , IC )
         XSAR( KC ) = SVOLM( IC )
  150 CONTINUE
C
      DO 170 KC = 1 , NOFVEC( INC )
         IC = KC + NC1 - 1
```

```
          RRN( KC ) = RRR( KC )
          URN( KC ) = RRR( KC ) * UUR( KC )
          VRN( KC ) = RRR( KC ) * VVR( KC )
          WRN( KC ) = RRR( KC ) * WWR( KC )
          EPN( KC ) = HYDV( IC , 8 ) + .5 * RRR( KC ) *
     .                       ( UUR( KC ) * UUR( KC ) +
     .                         VVR( KC ) * VVR( KC ) +
     .                         WWR( KC ) * WWR( KC ) )
          ARN( KC ) = RRR( KC ) * AAR( KC )
  170   CONTINUE
C --- COMPUTE THE SOURCE TERM FOR AXI SYMMETRY FLOW PROBLEM
C --- IF THE FLOW IS NOT AXI SYMMETRY , COMMENT LOOP 160
C
        DO 190 KC = 1 , NOFVEC( INC )
           IC = KC + NC1 - 1
        DTA = DTT * XSAR( KC )
C
        RRLL = DTA * RRL( KC )
        UULL = DTA * UUL( KC )
        VVLL = DTA * VVL( KC )
        WWLL = DTA * WWL( KC )
        RRN( KC ) = RRN( KC ) - RRLL
C
        URN( KC ) = URN( KC ) - UULL
C
        VRN( KC ) = VRN( KC ) - VVLL
C
        WRN( KC ) = WRN( KC ) - WWLL
C
        PPLL = DTA * PPL( KC )
        EPN( KC ) = EPN( KC ) - PPLL
C
        AALL = DTA * AAL( KC )
        ARN( KC ) = ARN( KC ) - AALL
C
  190   CONTINUE
C
        DO 195 IC = NC1 , NC2
           KC = IC - NC1 + 1
        HDUM            = 1. / RRN( KC )
        HYDV( IC , 1 ) = RRN( KC )
        HYDV( IC , 2 ) = URN( KC ) * HDUM
        HYDV( IC , 3 ) = VRN( KC ) * HDUM
        HYDV( IC , 4 ) = WRN( KC ) * HDUM
        HYDV( IC , 6 ) = ARN( KC ) * HDUM
  195   CONTINUE
C
        DO 200 IC = NC1 , NC2
           KC = IC - NC1 + 1
        HYDV( IC , 8 ) = ( EPN( KC )  - .5 * HYDV( IC , 1 ) *
     .                   ( HYDV( IC , 2 ) * HYDV( IC , 2 ) +
     .                     HYDV( IC , 3 ) * HYDV( IC , 3 ) +
     .                     HYDV( IC , 4 ) * HYDV( IC , 4 ) ) )
  200   CONTINUE
C
       IF( IEOS . EQ . 1 ) THEN
        TLIMIT = .9
        ITER = 6
        DO IC = NC1 , NC2
           KC = IC - NC1 + 1
C
       NITER = 0
       IF( HYDV( IC , 6 ) . LE . .2 ) THEN
C
       DST = HYDV( IC , 1 ) * GPERCC
       VOL = WMA * ( 1. - DST / FSA ) / DST / XGA
       EMEO = HYDV( IC , 8 ) / HYDV( IC , 1 ) * WMA / RGAS
C
       IYY = ( EMEO - EMEOA( 3 ) ) / RANGEA + 1
       IYY = MAXO( 1 , MINO( IYY , 47 ) )
C
       K = IYY + 2
       IYY = IYY
     . + INT( AMAX1( EMEO - EMEOA( K ) , 0.) / DYA( K ) )
```

```
     . - INT( AMAX1( EMEOA( K + 1 ) - EMEO , 0. ) / DYA( K ) )
      IYY = MAXO( 1, MINO( IYY , 47 ) )
C
      K1 = IYY + 2
      K2 = K1 + 1
      RT = ( EMEO - EMEOA( K1 ) ) / ( EMEOA( K2 ) - EMEOA( K1 ) )
      T = TA( K1 ) + 100. * RT
      CVM = CVMA( K1 ) + RT * ( CVMA( K2 ) - CVMA( K1 ) )
      ERS = 0.
C
      P = RGAS * T / VOL / GPERCC
      RGAMM1 = CVM
      HYDV( IC , 7 ) = 1. / RGAMM1 + 1.
      HYDV( IC , 5 ) = P
C
      ELSE
C
      DST = HYDV( IC , 1 ) * GPERCC
      VOL = WMX * ( 1. - DST / FSX ) / DST / XGX
      EMEO = HYDV( IC , 8 ) / HYDV( IC , 1 ) * WMX / RGAS
C
      IYY = ( EMEO - EMEOX( 3 ) ) / RANGEX + 1
      IYY = MAXO( 1 , MINO( IYY , 47 ) )
C
      K = IYY + 2
      IYY = IYY
     . + INT( AMAX1( EMEO - EMEOX( K ) , 0.) / DYX( K ) )
     . - INT( AMAX1( EMEOX( K + 1 ) - EMEO , 0. ) / DYX( K ) )
      IYY = MAXO( 1, MINO( IYY , 47 ) )
C
      K1 = IYY + 2
      K2 = K1 + 1
      RT = ( EMEO - EMEOX( K1 ) ) / ( EMEOX( K2 ) - EMEOX( K1 ) )
      T = TX( K1 ) + 100. * RT
      CVM = CVMX( K1 ) + RT * ( CVMX( K2 ) - CVMX( K1 ) )
      ERS = 0.
C
   10 CONTINUE
      P = RGAS * T / VOL / GPERCC
      RGAMM1 = CVM
C
      X = COVX / VOL / ( ( T + THETAX ) ** ALFAX )
      Z = X * EXP( BETAX * X )
      X = 1. + BETAX * X
      RT = ALFAX * T / ( T + THETAX )
      ERS = ERS + RT * Z * T
C
      IF ( ITER .EQ. NITER ) GO TO 20
C
      CVM = CVM * XGX + SCVX
     *            + RT * Z * ( 2. - RT / ALFAX - RT * X )
      T = T - AMIN1( ERS / CVM , TLIMIT * T )
C
      NITER = NITER + 1
C
      RT = 0.01 * T
      K1 = RT
      K1 = MINO ( K1, 49 )
      K1 = MAXO ( K1, 3  )
      K2 = K1 + 1
      RT = RT - K1
      CVM = CVMX(K1) + RT * ( CVMX( K2 ) - CVMX( K1 ) )
      ERS = EMEOX( K1 ) + RT * ( EMEOX( K2 ) - EMEOX( K1 ) )
      ERS = ERS - EMEO
C
      GO TO 10
C
   20 CONTINUE
      P = P * ( 1. + Z )
      RGAMM1 = ( RGAMM1 +
     *          RT * Z * ( 2. - RT / ALFAX - RT * X ) ) / ( 1. + Z )
      X = X * Z / ( 1. + Z )
      RGAMM1 = RGAMM1 / ( ( 1. - RT * X ) ** 2 + X * RGAMM1 )
      ERS = ERS / EMEO
```

```
            HYDV( IC , 7 ) = 1. / RGAMM1 + 1.
            HYDV( IC , 5 ) = P
            END IF
            END DO
C
            ELSE
C
             DO IC = NC1 , NC2
             HYDV( IC , 5 ) = HYDV( IC , 8 ) * ( HYDV( IC , 7 ) - 1. )
            END DO
            END IF
C
            NC1 = NC2 + 1
            NC2 = NC2 + NOFVEC( INC + 1 )
  110        CONTINUE
C
             IF( NPRTCL . NE . 0 ) CALL KYDRFL
             IJKNUM = IJKNUM + 1
             WRITE(10,*) TT,(PR(KKJJ),KKJJ=1,NPRTCL)
  140        CONTINUE
C
              PMAX = -10000000.
              DO 415 IC = 1 , NC
              IV1 = JC( 1 , IC )
              IV2 = JC( 2 , IC )
              IV3 = JC( 3 , IC )
              IV4 = JC( 4 , IC )
              HNUMM = HYDV( IC , 5 )
              HNUMN = XC( 4 , IC )
              HNUM( IV1 , 5 ) = HNUM( IV1 , 5 ) + HNUMM * HNUMN
              HNUM( IV1 , 1 ) = HNUM( IV1 , 1 ) + HNUMN
              HNUM( IV2 , 5 ) = HNUM( IV2 , 5 ) + HNUMM * HNUMN
              HNUM( IV2 , 1 ) = HNUM( IV2 , 1 ) + HNUMN
              HNUM( IV3 , 5 ) = HNUM( IV3 , 5 ) + HNUMM * HNUMN
              HNUM( IV3 , 1 ) = HNUM( IV3 , 1 ) + HNUMN
              HNUM( IV4 , 5 ) = HNUM( IV4 , 5 ) + HNUMM * HNUMN
              HNUM( IV4 , 1 ) = HNUM( IV4 , 1 ) + HNUMN
  415         CONTINUE
              DO IV = 1 , NV
              HNUM( IV , 5 ) = HNUM( IV , 5 ) / HNUM( IV , 1 )
              END DO
              DO IV = 1 , NV
              IF( HNUM( IV , 5 ) .GT. PMAX ) PMAX = HNUM( IV , 5 )
              END DO
              PRINT * , PMAX
C
              ISNS = 0
              DO 300 IS = 1 , NS
                 IF(JS(9,IS).EQ.6.AND.XS(2,IS).LT.1.9649) THEN
                 ISNS=ISNS+1
                 ISURF(ISNS)=IS
                 END IF
  300         CONTINUE
              print*,ISNS
C
C              STEVE FORMAT
C
              DO 312 IV = 1 , NV
              WRITE(17,1001) IV,(XV(KK,IV),KK=1,3)
 1001         FORMAT('n,',I5,',',2(F10.5,','),F10.5)
  312         CONTINUE
C
              DO 322 IS = 1 , ISNS
                 IK=ISURF(IS)
                 WRITE(18,1002) IS,(JS(KK,IK),KK=1,3),JS(3,IK)
 1002         FORMAT('en,',4(I10,','),I10)
  322         CONTINUE
C
              WRITE(19,1005) TT
 1005         FORMAT('time,',E13.5)
              ITWO = 1
              IZERO = 0
              DO 342 IS = 1 , ISNS
                 IK=ISURF(IS)
```

```
            WRITE(19,1003) IS,ITWO,IZERO,HNUM(JS(1,IK),5),HNUM(JS(2,IK),5),
     .                     HNUM(JS(3,IK),5),HNUM(JS(3,IK),5)
 1003       FORMAT('sfe.',2(I5,','),'pres.',I5,',',3(E12.5,','),E12.5)
 342        CONTINUE
C
            WRITE(14,10101) 3*ISNS,ISNS,NDUMMY1,NDUMMY3,NDUMMY3
 10101      FORMAT(5I8)
 10102      FORMAT(I8,3E20.7)
 10103      FORMAT(2I8,A6,3I8)
 10104      FORMAT(I8,E20.7)
            CALL RYDRFL
            KKVV = 0
            DO 310 IV = 1 , ISNS
               IK=ISURF(IV)
               IV1 = JS(1,IK)
               IV2 = JS(2,IK)
               IV3 = JS(3,IK)
               XXV = XV(1,IV1)
               YYV = XV(2,IV1)
               ZZV = XV(3,IV1)
               XNN = -XN(IK)
               YNN = -YN(IK)
               ZNN = -ZN(IK)
               XXX = XXV + XNN * .001
               YYY = YYV + YNN * .001
               ZZZ = ZZV + ZNN * .001
               KKVV = KKVV + 1
               WRITE(14,10102) KKVV,XXX,YYY,ZZZ
               XXV = XV(1,IV2)
               YYV = XV(2,IV2)
               ZZV = XV(3,IV2)
               XXX = XXV + XNN * .001
               YYY = YYV + YNN * .001
               ZZZ = ZZV + ZNN * .001
               KKVV = KKVV + 1
               WRITE(14,10102) KKVV,XXX,YYY,ZZZ
               XXV = XV(1,IV3)
               YYV = XV(2,IV3)
               ZZV = XV(3,IV3)
               XXX = XXV + XNN * .001
               YYY = YYV + YNN * .001
               ZZZ = ZZV + ZNN * .001
               KKVV = KKVV + 1
               WRITE(14,10102) KKVV,XXX,YYY,ZZZ
 310        CONTINUE
            KKVV = 0
            DO 320 IS = 1 , ISNS
               IK=ISURF(IS)
               WRITE(14,10103) IS,IS,CTRI,KKVV+1,KKVV+2,KKVV+3
               KKVV = KKVV + 3
 320        CONTINUE
            WRITE(14,10101) VDATA
            WRITE(14,*) VLABEL
            KKVV = 0
            DO 430 IV = 1 , ISNS
               IK=ISURF(IV)
               PRR = PR(IK)
               WRITE(14,10104) KKVV+1,PRR
               WRITE(14,10104) KKVV+2,PRR
               WRITE(14,10104) KKVV+3,PRR
               KKVV = KKVV + 3
 430        CONTINUE
            ISNS = 0
            DO IS = 1 , NS
               IF(JS(9,IS).EQ.6) THEN
               XXS = XS(1,IS)
               YYS = XS(2,IS)
               ZZS = XS(3,IS)
               ISNS=ISNS+1
               ISURF(ISNS)=IS
               END IF
            END DO
            print*,ISNS
            WRITE(15,10101) 3*ISNS,ISNS,NDUMMY1,NDUMMY3,NDUMMY3
```

```
      KKVV = 0
      DO 410 IV = 1 , ISNS
         IK=ISURF(IV)
         IV1 = JS(1,IK)
         IV2 = JS(2,IK)
         IV3 = JS(3,IK)
         XNN = -XN(IK)
         YNN = -YN(IK)
         ZNN = -ZN(IK)
         XXV = XV(1,IV1)
         YYV = XV(2,IV1)
         ZZV = XV(3,IV1)
         XXX = XXV + XNN * .001
         YYY = YYV + YNN * .001
         ZZZ = ZZV + ZNN * .001
         KKVV = KKVV + 1
         WRITE(15,10102) KKVV,XXX,YYY,ZZZ
         XXV = XV(1,IV2)
         YYV = XV(2,IV2)
         ZZV = XV(3,IV2)
         XXX = XXV + XNN * .001
         YYY = YYV + YNN * .001
         ZZZ = ZZV + ZNN * .001
         KKVV = KKVV + 1
         WRITE(15,10102) KKVV,XXX,YYY,ZZZ
         XXV = XV(1,IV3)
         YYV = XV(2,IV3)
         ZZV = XV(3,IV3)
         XXX = XXV + XNN * .001
         YYY = YYV + YNN * .001
         ZZZ = ZZV + ZNN * .001
         KKVV = KKVV + 1
         WRITE(15,10102) KKVV,XXX,YYY,ZZZ
  410 CONTINUE
      KKVV = 0
      DO 420 IS = 1 , ISNS
         IK=ISURF(IS)
         WRITE(15,10103) IS,IS,CTRI,KKVV+1,KKVV+2,KKVV+3
         KKVV = KKVV + 3
  420 CONTINUE
      WRITE(15,10101) VDATA
      WRITE(15,*) VLABEL
      KKVV = 0
      DO 330 IV = 1 , ISNS
         IK=ISURF(IV)
         PRR = PR(IK)
         WRITE(15,10104) KKVV+1,PRR
         WRITE(15,10104) KKVV+2,PRR
         WRITE(15,10104) KKVV+3,PRR
         KKVV = KKVV + 3
  330 CONTINUE
C
C
C=======================================================================
C
C             I--------------------------------I
C             I  OUTPUT FILE FOR RESTARTS      I
C             I--------------------------------I
C
      IF( JT . EQ . 1 ) THEN
      REWIND 8
      WRITE(9) NV,NE,NS,NC,NTIME
      WRITE(9) ((XV(IK,IV),IK=1,3),IV=1,NV)
      WRITE(9) ((JE(KK,IE),KK=1,2),IE=1,NE)
      WRITE(9) ((JS(KK,IS),KK=1,9),(XS(KI,IS),KI=1,5),
     .             XN(IS),YN(IS),ZN(IS),XP(IS),YP(IS),ZP(IS),
     .             XT(IS),YT(IS),ZT(IS),IS=1,NS)
      WRITE(9) ((XYZMDL(KI,IS),KI=1,4),IS=1,NS)
      WRITE(9) ((JC(KK,IC),KK=1,8),(XC(KI,IC),KI=1,4),IC=1,NC)
      WRITE(9) ((RGRAD(IC,KI),UGRAD(IC,KI),VGRAD(IC,KI),
     .             WGRAD(IC,KI),PGRAD(IC,KI),KI=1,3),IC=1,NC)
      WRITE(9) SAREVG,
     .           NVECE,NREME,NVECV,NREMV,NVECS,NREMS,NVECC,NREMC
      WRITE(9) RIN,PIN,RINL,PINL,UVIN,UIN,VIN,WIN,TT
```

```
            WRITE(9) NPRTCL
            IF(NPRTCL.GT.0)
     .      WRITE(9) (IJKPRT(IK),IK=1,NPRTCL)
            END IF
            WRITE(9) ((HYDV(IC,IK),IK=1,8),IC=1,NC)
C
            REWIND 88
            WRITE(88) NV,NE,NS,NC,NTIME
            WRITE(88) ((XV(IK,IV),IK=1,3),IV=1,NV)
            WRITE(88) ((JE(KK,IE),KK=1,2),IE=1,NE)
            WRITE(88) ((JS(KK,IS),KK=1,9),(XS(KI,IS),KI=1,5),
     .                 XN(IS),YN(IS),ZN(IS),XP(IS),YP(IS),ZP(IS),
     .                 XT(IS),YT(IS),ZT(IS),IS=1,NS)
            WRITE(88) ((XYZMDL(KI,IS),KI=1,4),IS=1,NS)
            WRITE(88) ((JC(KK,IC),KK=1,8),(XC(KI,IC),KI=1,4),IC=1,NC)
            WRITE(88) ((RGRAD(IC,KI),UGRAD(IC,KI),VGRAD(IC,KI),
     .                 WGRAD(IC,KI),PGRAD(IC,KI),KI=1,3),IC=1,NC)
            WRITE(88) SAREVG,
     .                NVECE,NREME,NVECV,NREMV,NVECS,NREMS,NVECC,NREMC
            WRITE(88) RIN,PIN,RINL,PINL,UVIN,UIN,VIN,WIN,TT
            WRITE(88) NPRTCL
            IF(NPRTCL.GT.0)
     .      WRITE(88) (IJKPRT(IK),IK=1,NPRTCL)
            WRITE(88) ((HYDV(IC,IK),IK=1,8),IC=1,NC)
C
  120     CONTINUE
          REWIND 10
          REWIND 26
          WRITE(26,*) IJKNUM
          DO KKJ = 1 , IJKNUM
          READ (10,*) RO,(RRN(IK),IK=1,NPRTCL)
          WRITE (26,*) RO,(RRN(IK),IK=1,NPRTCL)
          END DO
C
        RETURN
        END
C
```

Thu Jul 1 14:17:00 1993 threed.f SUBROUTINE GEOMTR

```
        SUBROUTINE GEOMTR
C
C--------------------------------------------------------------I
C                                                              I
C     GEOMTR COMPUTE THE DUAL MESH AFTER INITIALIZATION THE GRID I
C                                                              I
C--------------------------------------------------------------I
C
        include     'dmsh00.h'
        include     'dhydm0.h'
        include     'dphsm0.h'
        include     'dmtrl0.h'
C
C --- DEFINING BOUNDARY EDGES AND COMPUTING BAR CENTER OF TRIANGLES
C
        PRINT * , NE,NS
        DO 110 IC = 1 , NC
        SVOLM( IC ) = 1. / XC( 4 , IC )
  110   CONTINUE
C
        DO 120 IS = 1 , NS
C
        ICL = JS( 7 , IS )
        ICR = JS( 8 , IS )
C
        IV1 = JS( 1 , IS )
        IV2 = JS( 2 , IS )
        IV3 = JS( 3 , IS )
C
        X1 = XV( 1 , IV1 )
        Y1 = XV( 2 , IV1 )
        Z1 = XV( 3 , IV1 )
C
        A = XN( IS )
```

```
            B = YN( IS )
            C = ZN( IS )
      C
            D = - ( A * X1 + B * Y1 + C * Z1 )
      C
            XCL = XC( 1 , ICL )
            YCL = XC( 2 , ICL )
            ZCL = XC( 3 , ICL )
      C
            DD = A * XCL + B * YCL + C * ZCL + D
      C
            IATRB = JS( 9 , IS )
            IF( IATRB . EQ . 0 ) THEN
      C
            XCR = XC( 1 , ICR )
            YCR = XC( 2 , ICR )
            ZCR = XC( 3 , ICR )
      C
            XX = XCR - XCL
            YY = YCR - YCL
            ZZ = ZCR - ZCL
      C
            DDD = A * XX + B * YY + C * ZZ
      C
            XYZ = - DD / DDD
            XYZMDL( 4 , IS ) = XYZ
            XYZMDL( 1 , IS ) = XCL + XYZ * XX
            XYZMDL( 2 , IS ) = YCL + XYZ * YY
            XYZMDL( 3 , IS ) = ZCL + XYZ * ZZ
            XS( 5 , IS ) = SQRT( XX * XX + YY * YY + ZZ * ZZ )
      C
            ELSE
      C
            XYZ = - DD
            XYZMDL( 1 , IS ) = XCL + XYZ * A
            XYZMDL( 2 , IS ) = YCL + XYZ * B
            XYZMDL( 3 , IS ) = ZCL + XYZ * C
            XS( 5 , IS ) = ABS( XYZ )
            XYZMDL( 4 , IS ) = 1.
      C
            END IF
      C
        120 CONTINUE
      C
            RETURN
            END
      C
                SUBROUTINE UPDATE
      C
      C-------------------------------------------------------------I
      C                                                             I
      C      UPDATE COMPUTE THE DUAL MESH AFTER INITIALIZATION THE GRID I
      C                                                             I
      C-------------------------------------------------------------I
      C
            include     'dmsh00.h'
            include     'dhydm0.h'
            include     'dphsm0.h'
            include     'dmtrl0.h'
      C
      C
      C      READ IN VERTEX INFORMATION
      C
               READ (16,*) NV,NE,NC,NS
               DO 1110 IK = 1 , NV
               READ (16,*) IJ,XV(1,IK),XV(2,IK),XV(3,IK)
               XXX =XV(1,IK) + 34.5
               YYY =XV(2,IK) - 65.75
               ZZZ =XV(3,IK) + 11.5
               XV(1,IK)=XXX*.0254
               XV(2,IK)=YYY*.0254
               XV(3,IK)=ZZZ*.0254
       1110    CONTINUE
               PRINT * , NV
```

```
C
C       READ IN EDGE INFORMATION ( EDGES OF TRIANGLES).
C
        DO 1120 IK = 1 , NE
        READ (16,*) IJ,JE(1,IK),JE(2,IK)
 1120   CONTINUE
        PRINT * , NE
C
C       READ IN CELL (TETRAHIDRAL) INFORMATION.
C
C       CELL INFORMATION, FOR EACH CELL FOUR VERTICES
C
        DO 1130 IK = 1 , NC
        READ (16,*) IJ,JC(1,IK),JC(2,IK),JC(3,IK),JC(4,IK)
 1130   CONTINUE
C
        DO 1200 IK = 1 , NC
        IV1 = JC( 1 , IK )
        IV2 = JC( 2 , IK )
        IV3 = JC( 3 , IK )
        IV4 = JC( 4 , IK )
C
C       SIDE INFORMATION, FOR EACH CELL CENTROID OF CELL
C
        XC( 1 , IK ) = ( XV( 1 , IV1 ) + XV( 1 , IV2 ) +
     .                   XV( 1 , IV3 ) + XV( 1 , IV4 ) ) * .25
        XC( 2 , IK ) = ( XV( 2 , IV1 ) + XV( 2 , IV2 ) +
     .                   XV( 2 , IV3 ) + XV( 2 , IV4 ) ) * .25
        XC( 3 , IK ) = ( XV( 3 , IV1 ) + XV( 3 , IV2 ) +
     .                   XV( 3 , IV3 ) + XV( 3 , IV4 ) ) * .25
C
C       SIDE INFORMATION, FOR EACH CELL VOLUME OF CELL
C
        XPIJ = XV( 1 , IV2 ) - XV( 1 , IV1 )
        YPIJ = XV( 2 , IV2 ) - XV( 2 , IV1 )
        ZPIJ = XV( 3 , IV2 ) - XV( 3 , IV1 )
C
        XPIK = XV( 1 , IV3 ) - XV( 1 , IV1 )
        YPIK = XV( 2 , IV3 ) - XV( 2 , IV1 )
        ZPIK = XV( 3 , IV3 ) - XV( 3 , IV1 )
C
        XNIK = YPIJ * ZPIK - ZPIJ * YPIK
        YNIK = ZPIJ * XPIK - XPIJ * ZPIK
        ZNIK = XPIJ * YPIK - YPIJ * XPIK
C
        XPIJ = XV( 1 , IV4 ) - XV( 1 , IV1 )
        YPIJ = XV( 2 , IV4 ) - XV( 2 , IV1 )
        ZPIJ = XV( 3 , IV4 ) - XV( 3 , IV1 )
C
        VOL = ( XNIK * XPIJ + YNIK * YPIJ +
     .                    ZNIK * ZPIJ ) / 6.
        XC( 4 , IK ) = VOL
        IF( VOL . LT . 0. ) PRINT *,IK,VOL
 1200   CONTINUE
        PRINT * , NC
C
C       READ IN SIDE (TRIANGLE) INFORMATION.
C
C       SIDE INFORMATION, FOR EACH FACE THREE VERTICES
C
        DO 1150 IK = 1 , NS
        READ (16,*) IJ,JS(1,IK),JS(2,IK),JS(3,IK)
 1150   CONTINUE
        PRINT * , NS,NC
C
C       SIDE INFORMATION, FOR EACH FACE THREE EDGES
C
        DO 1160 IK = 1 , NS
        READ (16,*) IJ,JS(4,IK),JS(5,IK),JS(6,IK)
 1160   CONTINUE
        PRINT * , NS,NC,NV
C
C       CELL INFORMATION, FOR EACH CELL FOUR EDGES
C
```

```
      DO 1140 IK = 1 , NC
      READ (16,*) IJ,JC5,IDIR1,JC6,IDIR2,
     .                   JC7,IDIR3,JC8,IDIR4
      JC(5,IK) = IABS( JC5 )
      JC(6,IK) = IABS( JC6 )
      JC(7,IK) = IABS( JC7 )
      JC(8,IK) = IABS( JC8 )
 1140 CONTINUE
      PRINT * , NS,NC,NV,NE
C
C     SIDE INFORMATION, FOR EACH FACE LEFT AND RIGHT TETREHEDRA
C
      DO 1170 IK = 1 , NS
      READ (16,*) IJ,JS(7,IK),JS(8,IK)
      JS( 9 , IK ) = 0
 1170 CONTINUE
      PRINT * , NC,NV,NE
C
C     SIDE INFORMATION, FOR EACH FACE BOUNDARY CONDITION
C
 1180 CONTINUE
      READ (16,*,END=1210) IJ,IDUMY,JS(9,IJ)
      GO TO 1180
 1210 CONTINUE
      PRINT * , NV,NE,NS,NC
C
      DO 1190 IK = 1 , NS
      IV1 = JS( 1 , IK )
      IV2 = JS( 2 , IK )
      IV3 = JS( 3 , IK )
C
C     SIDE INFORMATION, FOR EACH FACE TANGENTIAL VECTOR
C
      XP( IK ) = XV( 1 , IV2 ) - XV( 1 , IV1 )
      YP( IK ) = XV( 2 , IV2 ) - XV( 2 , IV1 )
      ZP( IK ) = XV( 3 , IV2 ) - XV( 3 , IV1 )
      XPDUMY = XV( 1 , IV3 ) - XV( 1 , IV1 )
      YPDUMY = XV( 2 , IV3 ) - XV( 2 , IV1 )
      ZPDUMY = XV( 3 , IV3 ) - XV( 3 , IV1 )
C
C     SIDE INFORMATION, FOR EACH FACE NORMAL UNIT VECTOR
C
      XN( IK ) = YP( IK ) * ZPDUMY - ZP( IK ) * YPDUMY
      YN( IK ) = ZP( IK ) * XPDUMY - XP( IK ) * ZPDUMY
      ZN( IK ) = XP( IK ) * YPDUMY - YP( IK ) * XPDUMY
C
C     SIDE INFORMATION, FOR EACH FACE TANGENTIAL VECTOR
C
      XT( IK ) = - YP( IK ) * ZN( IK ) + ZP( IK ) * YN( IK )
      YT( IK ) = - ZP( IK ) * XN( IK ) + XP( IK ) * ZN( IK )
      ZT( IK ) = - XP( IK ) * YN( IK ) + YP( IK ) * XN( IK )
C
      XYZDUM = XN(IK)*XN(IK) + YN(IK)*YN(IK) + ZN(IK)*ZN(IK)
      IF(XYZDUM.EQ.0.) PRINT *,IK
      XYZDUM = 1. / SQRT( XYZDUM )
C
C     SIDE INFORMATION, FOR EACH FACE AREA OF FACE
C
      XS( 4 , IK ) = .5 / XYZDUM
C
C     SIDE INFORMATION, FOR EACH FACE CENTROID OF FACE
C
      XS( 1 , IK ) = ( XV( 1 , IV1 ) + XV( 1 , IV2 ) +
     .                 XV( 1 , IV3 ) ) / 3.
      XS( 2 , IK ) = ( XV( 2 , IV1 ) + XV( 2 , IV2 ) +
     .                 XV( 2 , IV3 ) ) / 3.
      XS( 3 , IK ) = ( XV( 3 , IV1 ) + XV( 3 , IV2 ) +
     .                 XV( 3 , IV3 ) ) / 3.
      XN( IK ) = XN( IK ) * XYZDUM
      YN( IK ) = YN( IK ) * XYZDUM
      ZN( IK ) = ZN( IK ) * XYZDUM
      XYZDUM = XP(IK)*XP(IK) + YP(IK)*YP(IK) + ZP(IK)*ZP(IK)
      XYZDUM = 1. / SQRT( XYZDUM )
      XP( IK) = XP( IK ) * XYZDUM
```

```
              YP( IK) = YP( IK ) * XYZDUM
              ZP( IK) = ZP( IK ) * XYZDUM
              XYZDUM = XT(IK)*XT(IK) + YT(IK)*YT(IK) + ZT(IK)*ZT(IK)
              XYZDUM = 1. / SQRT( XYZDUM )
              XT( IK) = XT( IK ) * XYZDUM
              YT( IK) = YT( IK ) * XYZDUM
              ZT( IK) = ZT( IK ) * XYZDUM
 1190         CONTINUE
              PRINT * , NS
C
        NVECV = NV / 128
        NREMV = NV - NVECV * 128
        NVECE = NE / 128
        NREME = NE - NVECE * 128
        NVECS = NS / 128
        NREMS = NS - NVECS * 128
        NVECC = NC / 128
        NREMC = NC - NVECC * 128
C
        DO 125 INV = 1 , NVECV
        NOFVEV( INV ) = 128
  125   CONTINUE
        NVEEV = NVECV
        IF( NREMV . GT . 0 ) THEN
        NVEEV = NVECV + 1
        NOFVEV( NVEEV ) = NREMV
        END IF
C
        DO 105 INE = 1 , NVECE
        NOFVEE( INE ) = 128
  105   CONTINUE
        NVEEE = NVECE
        IF( NREME . GT . 0 ) THEN
        NVEEE = NVECE + 1
        NOFVEE( NVEEE ) = NREME
        END IF
C
        DO 115 INS = 1 , NVECS
        NOFVES( INS ) = 128
  115   CONTINUE
        NVEES = NVECS
        IF( NREMS . GT . 0 ) THEN
        NVEES = NVECS + 1
        NOFVES( NVEES ) = NREMS
        END IF
C
        DO 135 INC = 1 , NVECC
        NOFVEC( INC ) = 128
  135   CONTINUE
        NVEEC = NVECC
        IF( NREMC . GT . 0 ) THEN
        NVEEC = NVECC + 1
        NOFVEC( NVEEC ) = NREMC
        END IF
C
         PRINT *,NV,NE,NS,NC
         PRINT *,NVEEV,NVEEE,NVEES,NVEEC
         PRINT *,NREMV,NREME,NREMS,NREMC
 1001    FORMAT(4I7)
 1002    FORMAT(I7,3E20.12)
C
        CALL GEOMTR
C
        RETURN
        END
C
```

```
      SUBROUTINE UPGRAD
C
C----------------------------------------------------------------I
C                                                                I
C     UPGRAD COMPUTE THE DUAL MESH AFTER ADDAPTING THE GRID      I
C                                                                I
C----------------------------------------------------------------I
C
      include    'dmsh00.h'
      include    'dhydm0.h'
      include    'dphsm0.h'
      include    'dmtrl0.h'
C
      REAL XELEFT(128),YELEFT(128),XERIGT(128),YERIGT(128)
C
C --- DEFINING BOUNDARY EDGES
C
        READ(8) NV,NE,NS,NC,NTINE
        READ(8) ((XV(IK,IV),IK=1,3),IV=1,NV)
        READ(8) ((JE(KK,IE),KK=1,2),IE=1,NE)
        READ(8) ((JS(KK,IS),KK=1,9),(XS(KI,IS),KI=1,5),
     .           XN(IS),YN(IS),ZN(IS),XP(IS),YP(IS),ZP(IS),
     .           XT(IS),YT(IS),ZT(IS),IS=1,NS)
        READ(8) ((XYZMDL(KI,IS),KI=1,4),IS=1,NS)
        READ(8) ((JC(KK,IC),KK=1,8),(XC(KI,IC),KI=1,4),IC=1,NC)
        READ(8) ((RGRAD(IC,KI),UGRAD(IC,KI),VGRAD(IC,KI),
     .           WGRAD(IC,KI),PGRAD(IC,KI),KI=1,3),IC=1,NC)
        READ(8) SAREVG,
     .          NVECE,NREME,NVECV,NREMV,NVECS,NREMS,NVECC,NREMC
         PRINT * , NE,NS
C
      DO 100 IC = 1 , NC
      SVOLM( IC ) = 1. / XC( 4 , IC )
  100 CONTINUE
C
      DO 105 INE = 1 , NVECE
      NOFVEE( INE ) = 128
  105 CONTINUE
      NVEEE = NVECE
      IF( NREME . GT . 0 ) THEN
      NVEEE = NVECE + 1
      NOFVEE( NVEEE ) = NREME
      END IF
C
      DO 115 INS = 1 , NVECS
      NOFVES( INS ) = 128
  115 CONTINUE
      NVEES = NVECS
      IF( NREMS . GT . 0 ) THEN
      NVEES = NVECS + 1
      NOFVES( NVEES ) = NREMS
      END IF
C
      DO 125 INV = 1 , NVECV
      NOFVEV( INV ) = 128
  125 CONTINUE
      NVEEV = NVECV
      IF( NREMV . GT . 0 ) THEN
      NVEEV = NVECV + 1
      NOFVEV( NVEEV ) = NREMV
      END IF
C
      DO 135 INC = 1 , NVECC
      NOFVEC( INC ) = 128
  135 CONTINUE
      NVEEC = NVECC
      IF( NREMC . GT . 0 ) THEN
      NVEEC = NVECC + 1
      NOFVEC( NVEEC ) = NREMC
      END IF
C
       PRINT *,NV,NE,NS,NC,NVECV,NREMV,NVECE,NREME,NVECS,NREMS,
     .                          NVECC,NREMC
C
```

```
      RETURN
      END
C
```

Thu Jul 1 14:17:00 1993   threed.f   SUBROUTINE GRADNT

```
      SUBROUTINE GRADNT
C
C------------------------------------------------------------I
C                                                            I
C        GRADNT COMPUTE THE GRADIENT FOR SECOND ORDER CALCULATION    I
C                                                            I
C------------------------------------------------------------I
C
      include      'dmsh00.h'
      include      'dhydm0.h'
      include      'dphsm0.h'
      include      'dmtrl0.h'
C
      REAL RRMIDL(128),PPMIDL(128),UUMIDL(128),VVMIDL(128),
     .     WWMIDL(128),AAMIDL(128)
      REAL RIGRAD(128),PIGRAD(128),UIGRAD(128),VIGRAD(128),
     .     WIGRAD(128),AIGRAD(128)
      REAL RJGRAD(128),PJGRAD(128),UJGRAD(128),VJGRAD(128),
     .     WJGRAD(128),AJGRAD(128)
      REAL RKGRAD(128),PKGRAD(128),UKGRAD(128),VKGRAD(128),
     .     WKGRAD(128),AKGRAD(128)
      REAL RMAX(128),PMAX(128),UMAX(128),VMAX(128),WMAX(128),
     .     AMAX(128)
      REAL RMIN(128),PMIN(128),UMIN(128),VMIN(128),WMIN(128),
     .     AMIN(128)
      REAL ROR(4),UOR(4),VOR(4),WOR(4),POR(4),AOR(4)
      REAL ROL(4),UOL(4),VOL(4),WOL(4),POL(4),AOL(4)
C
      DO 120 IH = 1 , 3
C
      DO 120 IC = 1 , NC
C
      RGRAD( IC , IH ) = 0.
      UGRAD( IC , IH ) = 0.
      VGRAD( IC , IH ) = 0.
      WGRAD( IC , IH ) = 0.
      PGRAD( IC , IH ) = 0.
C
  120 CONTINUE
C
C --- BEGIN LOOP OVER ALL EDGES IN THE DOMAIN ----------------------
C
      NS1 = 1
      NS2 = NOFVES( 1 )
      DO 90 INS = 1 , NVEES
C
C --- FETCH HYDRO QUANTITIES ------------------------------------
C
      DO 105 IS = NS1 , NS2
         KS = IS - NS1 + 1
C
       ICL = JS( 7 , IS )
       ICR = JS( 8 , IS )
C
      IATRB = JS( 9 , IS )
      IF( IATRB . EQ . 0 ) THEN
C
       XYZ = XYZMDL( 4 , IS )
         RRMIDL( KS ) = HYDV( ICL , 1 ) + XYZ * ( HYDV( ICR , 1 ) -
     .                                            HYDV( ICL , 1 ) )
         UUMIDL( KS ) = HYDV( ICL , 2 ) + XYZ * ( HYDV( ICR , 2 ) -
     .                                            HYDV( ICL , 2 ) )
         VVMIDL( KS ) = HYDV( ICL , 3 ) + XYZ * ( HYDV( ICR , 3 ) -
     .                                            HYDV( ICL , 3 ) )
         WWMIDL( KS ) = HYDV( ICL , 4 ) + XYZ * ( HYDV( ICR , 4 ) -
     .                                            HYDV( ICL , 4 ) )
         PPMIDL( KS ) = HYDV( ICL , 5 ) + XYZ * ( HYDV( ICR , 5 ) -
     .                                            HYDV( ICL , 5 ) )
```

```
C
       ELSE
C
          RRMIDL( KS ) = HYDV( ICL , 1 )
          UUMIDL( KS ) = HYDV( ICL , 2 )
          VVMIDL( KS ) = HYDV( ICL , 3 )
          WWMIDL( KS ) = HYDV( ICL , 4 )
          PPMIDL( KS ) = HYDV( ICL , 5 )
C
       END IF
C
  105  CONTINUE
C
       DO 110 IS = NS1 , NS2
          KS = IS - NS1 + 1
C
          XEXN = XS( 4 , IS ) * XN( IS )
          XEYN = XS( 4 , IS ) * YN( IS )
          XEZN = XS( 4 , IS ) * ZN( IS )
C
          RIGRAD( KS ) = RRMIDL( KS ) * XEXN
          UIGRAD( KS ) = UUMIDL( KS ) * XEXN
          VIGRAD( KS ) = VVMIDL( KS ) * XEXN
          WIGRAD( KS ) = WWMIDL( KS ) * XEXN
          PIGRAD( KS ) = PPMIDL( KS ) * XEXN
C
          RJGRAD( KS ) = RRMIDL( KS ) * XEYN
          UJGRAD( KS ) = UUMIDL( KS ) * XEYN
          VJGRAD( KS ) = VVMIDL( KS ) * XEYN
          WJGRAD( KS ) = WWMIDL( KS ) * XEYN
          PJGRAD( KS ) = PPMIDL( KS ) * XEYN
C
          RKGRAD( KS ) = RRMIDL( KS ) * XEZN
          UKGRAD( KS ) = UUMIDL( KS ) * XEZN
          VKGRAD( KS ) = VVMIDL( KS ) * XEZN
          WKGRAD( KS ) = WWMIDL( KS ) * XEZN
          PKGRAD( KS ) = PPMIDL( KS ) * XEZN
C
  110  CONTINUE
C
       DO 130 IS = NS1 , NS2
          KS = IS - NS1 + 1
C
        ICL = JS( 7 , IS )
        ICR = JS( 8 , IS )
C
       RGRAD( ICL , 1 ) = RGRAD( ICL , 1 ) + RIGRAD( KS )
       RGRAD( ICL , 2 ) = RGRAD( ICL , 2 ) + RJGRAD( KS )
       RGRAD( ICL , 3 ) = RGRAD( ICL , 3 ) + RKGRAD( KS )
       UGRAD( ICL , 1 ) = UGRAD( ICL , 1 ) + UIGRAD( KS )
       UGRAD( ICL , 2 ) = UGRAD( ICL , 2 ) + UJGRAD( KS )
       UGRAD( ICL , 3 ) = UGRAD( ICL , 3 ) + UKGRAD( KS )
       VGRAD( ICL , 1 ) = VGRAD( ICL , 1 ) + VIGRAD( KS )
       VGRAD( ICL , 2 ) = VGRAD( ICL , 2 ) + VJGRAD( KS )
       VGRAD( ICL , 3 ) = VGRAD( ICL , 3 ) + VKGRAD( KS )
       WGRAD( ICL , 1 ) = WGRAD( ICL , 1 ) + WIGRAD( KS )
       WGRAD( ICL , 2 ) = WGRAD( ICL , 2 ) + WJGRAD( KS )
       WGRAD( ICL , 3 ) = WGRAD( ICL , 3 ) + WKGRAD( KS )
       PGRAD( ICL , 1 ) = PGRAD( ICL , 1 ) + PIGRAD( KS )
       PGRAD( ICL , 2 ) = PGRAD( ICL , 2 ) + PJGRAD( KS )
       PGRAD( ICL , 3 ) = PGRAD( ICL , 3 ) + PKGRAD( KS )
C
       IATRB = JS( 9 , IS )
       IF( IATRB . EQ . 0 ) THEN
C
C ... GRADIENT OF DENSITY ( U V W DIRECTION ) .......................
C
       RGRAD( ICR , 1 ) = RGRAD( ICR , 1 ) - RIGRAD( KS )
       RGRAD( ICR , 2 ) = RGRAD( ICR , 2 ) - RJGRAD( KS )
       RGRAD( ICR , 3 ) = RGRAD( ICR , 3 ) - RKGRAD( KS )
C
C ... GRADIENT OF U VELOCITY ( U V W DIRECTION ). ..................
C
       UGRAD( ICR , 1 ) = UGRAD( ICR , 1 ) - UIGRAD( KS )
```

```
            UGRAD( ICR , 2 ) = UGRAD( ICR , 2 ) - UJGRAD( KS )
            UGRAD( ICR , 3 ) = UGRAD( ICR , 3 ) - UKGRAD( KS )
C
C ... GRADIENT OF V VELOCITY ( U V W DIRECTION ) ....................
C
            VGRAD( ICR , 1 ) = VGRAD( ICR , 1 ) - VIGRAD( KS )
            VGRAD( ICR , 2 ) = VGRAD( ICR , 2 ) - VJGRAD( KS )
            VGRAD( ICR , 3 ) = VGRAD( ICR , 3 ) - VKGRAD( KS )
C
C ... GRADIENT OF W VELOCITY ( U V W DIRECTION ) ....................
C
            WGRAD( ICR , 1 ) = WGRAD( ICR , 1 ) - WIGRAD( KS )
            WGRAD( ICR , 2 ) = WGRAD( ICR , 2 ) - WJGRAD( KS )
            WGRAD( ICR , 3 ) = WGRAD( ICR , 3 ) - WKGRAD( KS )
C
C ... GRADIENT OF PRESSURE ( U V W DIRECTION ) ...................  .
C
            PGRAD( ICR , 1 ) = PGRAD( ICR , 1 ) - PIGRAD( KS )
            PGRAD( ICR , 2 ) = PGRAD( ICR , 2 ) - PJGRAD( KS )
            PGRAD( ICR , 3 ) = PGRAD( ICR , 3 ) - PKGRAD( KS )
C
            END IF
C
  130       CONTINUE
C
            NS1 = NS2 + 1
            NS2 = NS2 + NOFVES( INS + 1 )
   90       CONTINUE
C
            DO 140 IH = 1 , 3
C
            DO 140 IC = 1 , NC
C
            RGRAD( IC , IH ) = RGRAD( IC , IH ) * SVOLM( IC )
            UGRAD( IC , IH ) = UGRAD( IC , IH ) * SVOLM( IC )
            VGRAD( IC , IH ) = VGRAD( IC , IH ) * SVOLM( IC )
            WGRAD( IC , IH ) = WGRAD( IC , IH ) * SVOLM( IC )
            PGRAD( IC , IH ) = PGRAD( IC , IH ) * SVOLM( IC )
C
  140       CONTINUE
C
            NC1 = 1
            NC2 = NOFVEC( 1 )
            DO 80 INC = 1 , NVEEC
C
            DO 150 IC = NC1 , NC2
                KC = IC - NC1 + 1
C
             IS = JC( 5 , IC )
C
             ICL = JS( 7 , IS )
             ICR = JS( 8 , IS )
C
             RROL = HYDV( ICL , 1 )
             UUOL = HYDV( ICL , 2 )
             VVOL = HYDV( ICL , 3 )
             WWOL = HYDV( ICL , 4 )
             PPOL = HYDV( ICL , 5 )
C
            IATRB = JS( 9 , IS )
            IF( IATRB . EQ . 0 ) THEN
C
             RROR = HYDV( ICR , 1 )
             UUOR = HYDV( ICR , 2 )
             VVOR = HYDV( ICR , 3 )
             WWOR = HYDV( ICR , 4 )
             PPOR = HYDV( ICR , 5 )
C
             ELSE
C
             RROR = RROL
             UUOR = UUOL
             VVOR = VVOL
             WWOR = WWOL
```

```fortran
            PPOR = PPOL
C
            END IF
C
            ROL( 1 ) = RROL
            UOL( 1 ) = UUOL
            VOL( 1 ) = VVOL
            WOL( 1 ) = WWOL
            POL( 1 ) = PPOL
C
            ROR( 1 ) = RROR
            UOR( 1 ) = UUOR
            VOR( 1 ) = VVOR
            WOR( 1 ) = WWOR
            POR( 1 ) = PPOR
C
            IS = JC( 6 , IC )
C
            ICL = JS( 7 , IS )
            ICR = JS( 8 , IS )
C
            RROL = HYDV( ICL , 1 )
            UUOL = HYDV( ICL , 2 )
            VVOL = HYDV( ICL , 3 )
            WWOL = HYDV( ICL , 4 )
            PPOL = HYDV( ICL , 5 )
C
          IATRB = JS( 9 , IS )
          IF( IATRB . EQ . 0 ) THEN
C
            RROR = HYDV( ICR , 1 )
            UUOR = HYDV( ICR , 2 )
            VVOR = HYDV( ICR , 3 )
            WWOR = HYDV( ICR , 4 )
            PPOR = HYDV( ICR , 5 )
C
            ELSE
C
            RROR = RROL
            UUOR = UUOL
            VVOR = VVOL
            WWOR = WWOL
            PPOR = PPOL
C
            END IF
C
            ROL( 2 ) = RROL
            UOL( 2 ) = UUOL
            VOL( 2 ) = VVOL
            WOL( 2 ) = WWOL
            POL( 2 ) = PPOL
C
            ROR( 2 ) = RROR
            UOR( 2 ) = UUOR
            VOR( 2 ) = VVOR
            WOR( 2 ) = WWOR
            POR( 2 ) = PPOR
C
            IS = JC( 7 , IC )
C
            ICL = JS( 7 , IS )
            ICR = JS( 8 , IS )
C
            RROL = HYDV( ICL , 1 )
            UUOL = HYDV( ICL , 2 )
            VVOL = HYDV( ICL , 3 )
            WWOL = HYDV( ICL , 4 )
            PPOL = HYDV( ICL , 5 )
C
          IATRB = JS( 9 , IS )
          IF( IATRB . EQ . 0 ) THEN
C
            RROR = HYDV( ICR , 1 )
            UUOR = HYDV( ICR , 2 )
```

```
              VVOR = HYDV( ICR , 3 )
              WWOR = HYDV( ICR , 4 )
              PPOR = HYDV( ICR , 5 )
C
              ELSE
C
              RROR = RROL
              UUOR = UUOL
              VVOR = VVOL
              WWOR = WWOL
              PPOR = PPOL
C
              END IF
C
              ROL( 3 ) = RROL
              UOL( 3 ) = UUOL
              VOL( 3 ) = VVOL
              WOL( 3 ) = WWOL
              POL( 3 ) = PPOL
C
              ROR( 3 ) = RROR
              UOR( 3 ) = UUOR
              VOR( 3 ) = VVOR
              WOR( 3 ) = WWOR
              POR( 3 ) = PPOR
C
              IS = JC( 8 , IC )
C
              ICL = JS( 7 , IS )
              ICR = JS( 8 , IS )
C
              RROL = HYDV( ICL , 1 )
              UUOL = HYDV( ICL , 2 )
              VVOL = HYDV( ICL , 3 )
              WWOL = HYDV( ICL , 4 )
              PPOL = HYDV( ICL , 5 )
C
             IATRB = JS( 9 , IS )
             IF( IATRB . EQ . 0 ) THEN
C
              RROR = HYDV( ICR , 1 )
              UUOR = HYDV( ICR , 2 )
              VVOR = HYDV( ICR , 3 )
              WWOR = HYDV( ICR , 4 )
              PPOR = HYDV( ICR , 5 )
C
              ELSE
C
              RROR = RROL
              UUOR = UUOL
              VVOR = VVOL
              WWOR = WWOL
              PPOR = PPOL
C
              END IF
C
              ROL( 4 ) = RROL
              UOL( 4 ) = UUOL
              VOL( 4 ) = VVOL
              WOL( 4 ) = WWOL
              POL( 4 ) = PPOL
C
              ROR( 4 ) = RROR
              UOR( 4 ) = UUOR
              VOR( 4 ) = VVOR
              WOR( 4 ) = WWOR
              POR( 4 ) = PPOR
C
              RMAX( KC ) = AMAX1( ROL( 1 ) , ROL( 2 ) , ROL( 3 ) , ROL( 4 ) ,
     .                            ROR( 1 ) , ROR( 2 ) , ROR( 3 ) , ROR( 4 ) )
              UMAX( KC ) = AMAX1( UOL( 1 ) , UOL( 2 ) , UOL( 3 ) , UOL( 4 ) ,
     .                            UOR( 1 ) , UOR( 2 ) , UOR( 3 ) , UOR( 4 ) )
              VMAX( KC ) = AMAX1( VOL( 1 ) , VOL( 2 ) , VOL( 3 ) , VOL( 4 ) ,
     .                            VOR( 1 ) , VOR( 2 ) , VOR( 3 ) , VOR( 4 ) )
```

```
      WMAX( KC ) = AMAX1( WOL( 1 ) , WOL( 2 ) , WOL( 3 ) , WOL( 4 ) ,
     .                    WOR( 1 ) , WOR( 2 ) , WOR( 3 ) , WOR( 4 ) )
      PMAX( KC ) = AMAX1( POL( 1 ) , POL( 2 ) , POL( 3 ) , POL( 4 ) ,
     .                    POR( 1 ) , POR( 2 ) , POR( 3 ) , POR( 4 ) )
C
      RMIN( KC ) = AMIN1( ROL( 1 ) , ROL( 2 ) , ROL( 3 ) , ROL( 4 ) ,
     .                    ROR( 1 ) , ROR( 2 ) , ROR( 3 ) , ROR( 4 ) )
      UMIN( KC ) = AMIN1( UOL( 1 ) , UOL( 2 ) , UOL( 3 ) , UOL( 4 ) ,
     .                    UOR( 1 ) , UOR( 2 ) , UOR( 3 ) , UOR( 4 ) )
      VMIN( KC ) = AMIN1( VOL( 1 ) , VOL( 2 ) , VOL( 3 ) , VOL( 4 ) ,
     .                    VOR( 1 ) , VOR( 2 ) , VOR( 3 ) , VOR( 4 ) )
      WMIN( KC ) = AMIN1( WOL( 1 ) , WOL( 2 ) , WOL( 3 ) , WOL( 4 ) ,
     .                    WOR( 1 ) , WOR( 2 ) , WOR( 3 ) , WOR( 4 ) )
      PMIN( KC ) = AMIN1( POL( 1 ) , POL( 2 ) , POL( 3 ) , POL( 4 ) ,
     .                    POR( 1 ) , POR( 2 ) , POR( 3 ) , POR( 4 ) )
C
  150 CONTINUE
C
      DO 180 IC = NC1 , NC2
         KC = IC - NC1 + 1
C
       RRR( KC ) = RMAX( KC ) - HYDV( IC , 1 )
       RRL( KC ) = RMIN( KC ) - HYDV( IC , 1 )
       UUR( KC ) = UMAX( KC ) - HYDV( IC , 2 )
       UUL( KC ) = UMIN( KC ) - HYDV( IC , 2 )
       VVR( KC ) = VMAX( KC ) - HYDV( IC , 3 )
       VVL( KC ) = VMIN( KC ) - HYDV( IC , 3 )
       WWR( KC ) = WMAX( KC ) - HYDV( IC , 4 )
       WWL( KC ) = WMIN( KC ) - HYDV( IC , 4 )
       PPR( KC ) = PMAX( KC ) - HYDV( IC , 5 )
       PPL( KC ) = PMIN( KC ) - HYDV( IC , 5 )
C
  180 CONTINUE
C
      DO 170 IC = NC1 , NC2
         KC = IC - NC1 + 1
C
       IS = JC( 5 , IC )
C
       ICL = JS( 7 , IS )
       ICR = JS( 8 , IS )
C
       XML = XYZMDL( 1 , IS ) - XC( 1 , ICL )
       YML = XYZMDL( 2 , IS ) - XC( 2 , ICL )
       ZML = XYZMDL( 3 , IS ) - XC( 3 , ICL )
C
       RROL = 1.E-16 + RGRAD( ICL , 1 ) * XML +
     .                 RGRAD( ICL , 2 ) * YML + RGRAD( ICL , 3 ) * ZML
       UUOL = 1.E-16 + UGRAD( ICL , 1 ) * XML +
     .                 UGRAD( ICL , 2 ) * YML + UGRAD( ICL , 3 ) * ZML
       VVOL = 1.E-16 + VGRAD( ICL , 1 ) * XML +
     .                 VGRAD( ICL , 2 ) * YML + VGRAD( ICL , 3 ) * ZML
       WWOL = 1.E-16 + WGRAD( ICL , 1 ) * XML +
     .                 WGRAD( ICL , 2 ) * YML + WGRAD( ICL , 3 ) * ZML
       PPOL = 1.E-16 + PGRAD( ICL , 1 ) * XML +
     .                 PGRAD( ICL , 2 ) * YML + PGRAD( ICL , 3 ) * ZML
C
      IATRB = JS( 9 , IS )
      IF( IATRB . EQ . 0 ) THEN
C
       XMR = XYZMDL( 1 , IS ) - XC( 1 , ICR )
       YMR = XYZMDL( 2 , IS ) - XC( 2 , ICR )
       ZMR = XYZMDL( 3 , IS ) - XC( 3 , ICR )
C
       RROR = 1.E-16 + RGRAD( ICR , 1 ) * XMR +
     .                 RGRAD( ICR , 2 ) * YMR + RGRAD( ICR , 3 ) * ZMR
       UUOR = 1.E-16 + UGRAD( ICR , 1 ) * XMR +
     .                 UGRAD( ICR , 2 ) * YMR + UGRAD( ICR , 3 ) * ZMR
       VVOR = 1.E-16 + VGRAD( ICR , 1 ) * XMR +
     .                 VGRAD( ICR , 2 ) * YMP + VGRAD( ICR , 3 ) * ZMR
       WWOR = 1.E-16 + WGRAD( ICR , 1 ) * XMR +
     .                 WGRAD( ICR , 2 ) * YMR + WGRAD( ICR , 3 ) * ZMR
       PPOR = 1.E-16 + PGRAD( ICR , 1 ) * XMR +
     .                 PGRAD( ICR , 2 ) * YMR + PGRAD( ICR , 3 ) * ZMR
```

```
C
      ELSE
C
      RROR = RROL
      UUOR = UUOL
      VVOR = VVOL
      WWOR = WWOL
      PPOR = PPOL
C
      END IF
C
      ROL( 1 ) = 1. / RROL
      UOL( 1 ) = 1. / UUOL
      VOL( 1 ) = 1. / VVOL
      WOL( 1 ) = 1. / WWOL
      POL( 1 ) = 1. / PPOL
C
      ROR( 1 ) = 1. / RROR
      UOR( 1 ) = 1. / UUOR
      VOR( 1 ) = 1. / VVOR
      WOR( 1 ) = 1. / WWOR
      POR( 1 ) = 1. / PPOR
C
      IS = JC( 6 , IC )
C
      ICL = JS( 7 , IS )
      ICR = JS( 8 , IS )
C
      XML = XYZMDL( 1 , IS ) - XC( 1 , ICL )
      YML = XYZMDL( 2 , IS ) - XC( 2 , ICL )
      ZML = XYZMDL( 3 , IS ) - XC( 3 , ICL )
C
      RROL = 1.E-16 + RGRAD( ICL , 1 ) * XML +
     .                RGRAD( ICL , 2 ) * YML + RGRAD( ICL , 3 ) * ZML
      UUOL = 1.E-16 + UGRAD( ICL , 1 ) * XML +
     .                UGRAD( ICL , 2 ) * YML + UGRAD( ICL , 3 ) * ZML
      VVOL = 1.E-16 + VGRAD( ICL , 1 ) * XML +
     .                VGRAD( ICL , 2 ) * YML + VGRAD( ICL , 3 ) * ZML
      WWOL = 1.E-16 + WGRAD( ICL , 1 ) * XML +
     .                WGRAD( ICL , 2 ) * YML + WGRAD( ICL , 3 ) * ZML
      PPOL = 1.E-16 + PGRAD( ICL , 1 ) * XML +
     .                PGRAD( ICL , 2 ) * YML + PGRAD( ICL , 3 ) * ZML
C
     IATRB = JS( 9 , IS )
     IF( IATRB . EQ . 0 ) THEN
C
      XMR = XYZMDL( 1 , IS ) - XC( 1 , ICR )
      YMR = XYZMDL( 2 , IS ) - XC( 2 , ICR )
      ZMR = XYZMDL( 3 , IS ) - XC( 3 , ICR )
C
      RROR = 1.E-16 + RGRAD( ICR , 1 ) * XMR +
     .                RGRAD( ICR , 2 ) * YMR + RGRAD( ICR , 3 ) * ZMR
      UUOR = 1.E-16 + UGRAD( ICR , 1 ) * XMR +
     .                UGRAD( ICR , 2 ) * YMR + UGRAD( ICR , 3 ) * ZMR
      VVOR = 1.E-16 + VGRAD( ICR , 1 ) * XMR +
     .                VGRAD( ICR , 2 ) * YMR + VGRAD( ICR , 3 ) * ZMR
      WWOR = 1.E-16 + WGRAD( ICR , 1 ) * XMR +
     .                WGRAD( ICR , 2 ) * YMR + WGRAD( ICR , 3 ) * ZMR
      PPOR = 1.E-16 + PGRAD( ICR , 1 ) * XMR +
     .                PGRAD( ICR , 2 ) * YMR + PGRAD( ICR , 3 ) * ZMR
C
      ELSE
C
      RROR = RROL
      UUOR = UUOL
      VVOR = VVOL
      WWOR = WWOL
      PPOR = PPOL
C
      END IF
C
      ROL( 2 ) = 1. / RROL
      UOL( 2 ) = 1. / UUOL
      VOL( 2 ) = 1. / VVOL
```

```
      WOL( 2 ) = 1. / WWOL
      POL( 2 ) = 1. / PPOL
C
      ROR( 2 ) = 1. / RROR
      UOR( 2 ) = 1. / UUOR
      VOR( 2 ) = 1. / VVOR
      WOR( 2 ) = 1. / WWOR
      POR( 2 ) = 1. / PPOR
C
      IS = JC( 7 , IC )
C
      ICL = JS( 7 , IS )
      ICR = JS( 8 , IS )
C
      XML = XYZMDL( 1 , IS ) - XC( 1 , ICL )
      YML = XYZMDL( 2 , IS ) - XC( 2 , ICL )
      ZML = XYZMDL( 3 , IS ) - XC( 3 , ICL )
C
      RROL = 1.E-16 + RGRAD( ICL , 1 ) * XML +
     .                RGRAD( ICL , 2 ) * YML + RGRAD( ICL , 3 ) * ZML
      UUOL = 1.E-16 + UGRAD( ICL , 1 ) * XML +
     .                UGRAD( ICL , 2 ) * YML + UGRAD( ICL , 3 ) * ZML
      VVOL = 1.E-16 + VGRAD( ICL , 1 ) * XML +
     .                VGRAD( ICL , 2 ) * YML + VGRAD( ICL , 3 ) * ZML
      WWOL = 1.E-16 + WGRAD( ICL , 1 ) * XML +
     .                WGRAD( ICL , 2 ) * YML + WGRAD( ICL , 3 ) * ZML
      PPOL = 1.E-16 + PGRAD( ICL , 1 ) * XML +
     .                PGRAD( ICL , 2 ) * YML + PGRAD( ICL , 3 ) * ZML
C
      IATRB = JS( 9 , IS )
      IF( IATRB . EQ . 0 ) THEN
C
        XMR = XYZMDL( 1 , IS ) - XC( 1 , ICR )
        YMR = XYZMDL( 2 , IS ) - XC( 2 , ICR )
        ZMR = XYZMDL( 3 , IS ) - XC( 3 , ICR )
C
        RROR = 1.E-16 + RGRAD( ICR , 1 ) * XMR +
     .                  RGRAD( ICR , 2 ) * YMR + RGRAD( ICR , 3 ) * ZMR
        UUOR = 1.E-16 + UGRAD( ICR , 1 ) * XMR +
     .                  UGRAD( ICR , 2 ) * YMR + UGRAD( ICR , 3 ) * ZMR
        VVOR = 1.E-16 + VGRAD( ICR , 1 ) * XMR +
     .                  VGRAD( ICR , 2 ) * YMR + VGRAD( ICR , 3 ) * ZMR
        WWOR = 1.E-16 + WGRAD( ICR , 1 ) * XMR +
     .                  WGRAD( ICR , 2 ) * YMR + WGRAD( ICR , 3 ) * ZMR
        PPOR = 1.E-16 + PGRAD( ICR , 1 ) * XMR +
     .                  PGRAD( ICR , 2 ) * YMR + PGRAD( ICR , 3 ) * ZMR
C
      ELSE
C
        RROR = RROL
        UUOR = UUOL
        VVOR = VVOL
        WWOR = WWOL
        PPOR = PPOL
C
      END IF
C
      ROL( 3 ) = 1. / RROL
      UOL( 3 ) = 1. / UUOL
      VOL( 3 ) = 1. / VVOL
      WOL( 3 ) = 1. / WWOL
      POL( 3 ) = 1. / PPOL
C
      ROR( 3 ) = 1. / RROR
      UOR( 3 ) = 1. / UUOR
      VOR( 3 ) = 1. / VVOR
      WOR( 3 ) = 1. / WWOR
      POR( 3 ) = 1. / PPOR
C
      IS = JC( 8 , IC )
C
      ICL = JS( 7 , IS )
      ICR = JS( 8 , IS )
C
```

```
          XML = XYZMDL( 1 , IS ) - XC( 1 , ICL )
          YML = XYZMDL( 2 , IS ) - XC( 2 , ICL )
          ZML = XYZMDL( 3 , IS ) - XC( 3 , ICL )
C
          RROL = 1.E-16 + RGRAD( ICL , 1 ) * XML +
     .                    RGRAD( ICL , 2 ) * YML + RGRAD( ICL , 3 ) * ZML
          UUOL = 1.E-16 + UGRAD( ICL , 1 ) * XML +
     .                    UGRAD( ICL , 2 ) * YML + UGRAD( ICL , 3 ) * ZML
          VVOL = 1.E-16 + VGRAD( ICL , 1 ) * XML +
     .                    VGRAD( ICL , 2 ) * YML + VGRAD( ICL , 3 ) * ZML
          WWOL = 1.E-16 + WGRAD( ICL , 1 ) * XML +
     .                    WGRAD( ICL , 2 ) * YML + WGRAD( ICL , 3 ) * ZML
          PPOL = 1.E-16 + PGRAD( ICL , 1 ) * XML +
     .                    PGRAD( ICL , 2 ) * YML + PGRAD( ICL , 3 ) * ZML
C
         IATRB = JS( 9 , IS )
         IF( IATRB . EQ . 0 ) THEN
C
          XMR = XYZMDL( 1 , IS ) - XC( 1 , ICR )
          YMR = XYZMDL( 2 , IS ) - XC( 2 , ICR )
          ZMR = XYZMDL( 3 , IS ) - XC( 3 , ICR )
C
          RROR = 1.E-16 + RGRAD( ICR , 1 ) * XMR +
     .                    RGRAD( ICR , 2 ) * YMR + RGRAD( ICR , 3 ) * ZMR
          UUOR = 1.E-16 + UGRAD( ICR , 1 ) * XMR +
     .                    UGRAD( ICR , 2 ) * YMR + UGRAD( ICR , 3 ) * ZMR
          VVOR = 1.E-16 + VGRAD( ICR , 1 ) * XMR +
     .                    VGRAD( ICR , 2 ) * YMR + VGRAD( ICR , 3 ) * ZMR
          WWOR = 1.E-16 + WGRAD( ICR , 1 ) * XMR +
     .                    WGRAD( ICR , 2 ) * YMR + WGRAD( ICR , 3 ) * ZMR
          PPOR = 1.E-16 + PGRAD( ICR , 1 ) * XMR +
     .                    PGRAD( ICR , 2 ) * YMR + PGRAD( ICR , 3 ) * ZMR
C
          ELSE
C
          RROR = RROL
          UUOR = UUOL
          VVOR = VVOL
          WWOR = WWOL
          PPOR = PPOL
C
          END IF
C
          ROL( 4 ) = 1. / RROL
          UOL( 4 ) = 1. / UUOL
          VOL( 4 ) = 1. / VVOL
          WOL( 4 ) = 1. / WWOL
          POL( 4 ) = 1. / PPOL
C
          ROR( 4 ) = 1. / RROR
          UOR( 4 ) = 1. / UUOR
          VOR( 4 ) = 1. / VVOR
          WOR( 4 ) = 1. / WWOR
          POR( 4 ) = 1. / PPOR
C
          ISNR = SIGN( 1. , ROR( 1 ) )
          ISNL = SIGN( 1. , ROL( 1 ) )
C
          TEMPR = ( 1 + ISNR ) * RRR( KC ) +
     .            ( 1 - ISNR ) * RRL( KC )
          RUVPR1 = 0.5 * TEMPR * ROR( 1 )
C
          TEMPL = ( 1 + ISNL ) * RRR( KC ) +
     .            ( 1 - ISNL ) * RRL( KC )
          RUVPL1 = 0.5 * TEMPL * ROL( 1 )
C
          ISNR = SIGN( 1. , ROR( 2 ) )
          ISNL = SIGN( 1. , ROL( 2 ) )
C
          TEMPR = ( 1 + ISNR ) * RRR( KC ) +
     .            ( 1 - ISNR ) * RRL( KC )
          RUVPR2 = 0.5 * TEMPR * ROR( 2 )
C
          TEMPL = ( 1 + ISNL ) * RRR( KC ) +
```

```
     .          ( 1 - ISNL ) * RRL( KC )
      RUVPL2 = 0.5 * TEMPL * ROL( 2 )
C
      ISNR = SIGN( 1. , ROR( 3 ) )
      ISNL = SIGN( 1. , ROL( 3 ) )
C
      TEMPR = ( 1 + ISNR ) * RRR( KC ) +
     .          ( 1 - ISNR ) * RRL( KC )
      RUVPR3 = 0.5 * TEMPR * ROR( 3 )
C
      TEMPL = ( 1 + ISNL ) * RRR( KC ) +
     .          ( 1 - ISNL ) * RRL( KC )
      RUVPL3 = 0.5 * TEMPL * ROL( 3 )
C
      ISNR = SIGN( 1. , ROR( 4 ) )
      ISNL = SIGN( 1. , ROL( 4 ) )
C
      TEMPR = ( 1 + ISNR ) * RRR( KC ) +
     .          ( 1 - ISNR ) * RRL( KC )
      RUVPR4 = 0.5 * TEMPR * ROR( 4 )
C
      TEMPL = ( 1 + ISNL ) * RRR( KC ) +
     .          ( 1 - ISNL ) * RRL( KC )
      RUVPL4 = 0.5 * TEMPL * ROL( 4 )
C
      RMIN( KC ) = AMIN1( 1. , RUVPR1 , RUVPL1 , RUVPR2 , RUVPL2 ,
     .                         RUVPR3 , RUVPL3 , RUVPR4 , RUVPL4 )
C
      ISNR = SIGN( 1. , UOR( 1 ) )
      ISNL = SIGN( 1. , UOL( 1 ) )
C
      TEMPR = ( 1 + ISNR ) * UUR( KC ) +
     .          ( 1 - ISNR ) * UUL( KC )
      RUVPR1 = 0.5 * TEMPR * UOR( 1 )
C
      TEMPL = ( 1 + ISNL ) * UUR( KC ) +
     .          ( 1 - ISNL ) * UUL( KC )
      RUVPL1 = 0.5 * TEMPL * UOL( 1 )
C
      ISNR = SIGN( 1. , UOR( 2 ) )
      ISNL = SIGN( 1. , UOL( 2 ) )
C
      TEMPR = ( 1 + ISNR ) * UUR( KC ) +
     .          ( 1 - ISNR ) * UUL( KC )
      RUVPR2 = 0.5 * TEMPR * UOR( 2 )
C
      TEMPL = ( 1 + ISNL ) * UUR( KC ) +
     .          ( 1 - ISNL ) * UUL( KC )
      RUVPL2 = 0.5 * TEMPL * UOL( 2 )
C
      ISNR = SIGN( 1. , UOR( 3 ) )
      ISNL = SIGN( 1. , UOL( 3 ) )
C
      TEMPR = ( 1 + ISNR ) * UUR( KC ) +
     .          ( 1 - ISNR ) * UUL( KC )
      RUVPR3 = 0.5 * TEMPR * UOR( 3 )
C
      TEMPL = ( 1 + ISNL ) * UUR( KC ) +
     .          ( 1 - ISNL ) * UUL( KC )
      RUVPL3 = 0.5 * TEMPL * UOL( 3 )
C
      ISNR = SIGN( 1. , UOR( 4 ) )
      ISNL = SIGN( 1. , UOL( 4 ) )
C
      TEMPR = ( 1 + ISNR ) * UUR( KC ) +
     .          ( 1 - ISNR ) * UUL( KC )
      RUVPR4 = 0.5 * TEMPR * UOR( 4 )
C
      TEMPL = ( 1 + ISNL ) * UUR( KC ) +
     .          ( 1 - ISNL ) * UUL( KC )
      RUVPL4 = 0.5 * TEMPL * UOL( 4 )
C
      UMIN( KC ) = AMIN1( 1. , RUVPR1 , RUVPL1 , RUVPR2 , RUVPL2 ,
     .                         RUVPR3 , RUVPL3 , RUVPR4 , RUVPL4 )
```

```
C
      ISNR = SIGN( 1. , VOR( 1 ) )
      ISNL = SIGN( 1. , VOL( 1 ) )
C
      TEMPR = ( 1 + ISNR ) * VVR( KC ) +
     .        ( 1 - ISNR ) * VVL( KC )
      RUVPR1 = 0.5 * TEMPR * VOR( 1 )
C
      TEMPL = ( 1 + ISNL ) * VVR( KC ) +
     .        ( 1 - ISNL ) * VVL( KC )
      RUVPL1 = 0.5 * TEMPL * VOL( 1 )
C
      ISNR = SIGN( 1. , VOR( 2 ) )
      ISNL = SIGN( 1. , VOL( 2 ) )
C
      TEMPR = ( 1 + ISNR ) * VVR( KC ) +
     .        ( 1 - ISNR ) * VVL( KC )
      RUVPR2 = 0.5 * TEMPR * VOR( 2 )
C
      TEMPL = ( 1 + ISNL ) * VVR( KC ) +
     .        ( 1 - ISNL ) * VVL( KC )
      RUVPL2 = 0.5 * TEMPL * VOL( 2 )
C
      ISNR = SIGN( 1. , VOR( 3 ) )
      ISNL = SIGN( 1. , VOL( 3 ) )
C
      TEMPR = ( 1 + ISNR ) * VVR( KC ) +
     .        ( 1 - ISNR ) * VVL( KC )
      RUVPR3 = 0.5 * TEMPR * VOR( 3 )
C
      TEMPL = ( 1 + ISNL ) * VVR( KC ) +
     .        ( 1 - ISNL ) * VVL( KC )
      RUVPL3 = 0.5 * TEMPL * VOL( 3 )
C
      ISNR = SIGN( 1. , VOR( 4 ) )
      ISNL = SIGN( 1. , VOL( 4 ) )
C
      TEMPR = ( 1 + ISNR ) * VVR( KC ) +
     .        ( 1 - ISNR ) * VVL( KC )
      RUVPR4 = 0.5 * TEMPR * VOR( 4 )
C
      TEMPL = ( 1 + ISNL ) * VVR( KC ) +
     .        ( 1 - ISNL ) * VVL( KC )
      RUVPL4 = 0.5 * TEMPL * VOL( 4 )
C
      VMIN( KC ) = AMIN1( 1. , RUVPR1 , RUVPL1 , RUVPR2 , RUVPL2 ,
     .                         RUVPR3 , RUVPL3 , RUVPR4 , RUVPL4 )
C
      ISNR = SIGN( 1. , WOR( 1 ) )
      ISNL = SIGN( 1. , WOL( 1 ) )
C
      TEMPR = ( 1 + ISNR ) * WWR( KC ) +
     .        ( 1 - ISNR ) * WWL( KC )
      RUVPR1 = 0.5 * TEMPR * WOR( 1 )
C
      TEMPL = ( 1 + ISNL ) * WWR( KC ) +
     .        ( 1 - ISNL ) * WWL( KC )
      RUVPL1 = 0.5 * TEMPL * WOL( 1 )
C
      ISNR = SIGN( 1. , WOR( 2 ) )
      ISNL = SIGN( 1. , WOL( 2 ) )
C
      TEMPR = ( 1 + ISNR ) * WWR( KC ) +
     .        ( 1 - ISNR ) * WWL( KC )
      RUVPR2 = 0.5 * TEMPR * WOR( 2 )
C
      TEMPL = ( 1 + ISNL ) * WWR( KC ) +
     .        ( 1 - ISNL ) * WWL( KC )
      RUVPL2 = 0.5 * TEMPL * WOL( 2 )
C
      ISNR = SIGN( 1. , WOR( 3 ) )
      ISNL = SIGN( 1. , WOL( 3 ) )
C
      TEMPR = ( 1 + ISNR ) * WWR( KC ) +
```

```
     .              ( 1 - ISNR ) * WWL( KC )
      RUVPR3 = 0.5 * TEMPR * WOR( 3 )
C
      TEMPL = ( 1 + ISNL ) * WWR( KC ) +
     .              ( 1 - ISNL ) * WWL( KC )
      RUVPL3 = 0.5 * TEMPL * WOL( 3 )
C
      ISNR = SIGN( 1. , WOR( 4 ) )
      ISNL = SIGN( 1. , WOL( 4 ) )
C
      TEMPR = ( 1 + ISNR ) * WWR( KC ) +
     .              ( 1 - ISNR ) * WWL( KC )
      RUVPR4 = 0.5 * TEMPR * WOR( 4 )
C
      TEMPL = ( 1 + ISNL ) * WWR( KC ) +
     .              ( 1 - ISNL ) * WWL( KC )
      RUVPL4 = 0.5 * TEMPL * WOL( 4 )
C
      WMIN( KC ) = AMIN1( 1. , RUVPR1 , RUVPL1 , RUVPR2 , RUVPL2 ,
     .                         RUVPR3 , RUVPL3 , RUVPR4 , RUVPL4 )
C
      ISNR = SIGN( 1. , POR( 1 ) )
      ISNL = SIGN( 1. , POL( 1 ) )
C
      TEMPR = ( 1 + ISNR ) * PPR( KC ) +
     .              ( 1 - ISNR ) * PPL( KC )
      RUVPR1 = 0.5 * TEMPR * POR( 1 )
C
      TEMPL = ( 1 + ISNL ) * PPR( KC ) +
     .              ( 1 - ISNL ) * PPL( KC )
      RUVPL1 = 0.5 * TEMPL * POL( 1 )
C
      ISNR = SIGN( 1. , POR( 2 ) )
      ISNL = SIGN( 1. , POL( 2 ) )
C
      TEMPR = ( 1 + ISNR ) * PPR( KC ) +
     .              ( 1 - ISNR ) * PPL( KC )
      RUVPR2 = 0.5 * TEMPR * POR( 2 )
C
      TEMPL = ( 1 + ISNL ) * PPR( KC ) +
     .              ( 1 - ISNL ) * PPL( KC )
      RUVPL2 = 0.5 * TEMPL * POL( 2 )
C
      ISNR = SIGN( 1. , POR( 3 ) )
      ISNL = SIGN( 1. , POL( 3 ) )
C
      TEMPR = ( 1 + ISNR ) * PPR( KC ) +
     .              ( 1 - ISNR ) * PPL( KC )
      RUVPR3 = 0.5 * TEMPR * POR( 3 )
C
      TEMPL = ( 1 + ISNL ) * PPR( KC ) +
     .              ( 1 - ISNL ) * PPL( KC )
      RUVPL3 = 0.5 * TEMPL * POL( 3 )
C
      ISNR = SIGN( 1. , POR( 4 ) )
      ISNL = SIGN( 1. , POL( 4 ) )
C
      TEMPR = ( 1 + ISNR ) * PPR( KC ) +
     .              ( 1 - ISNR ) * PPL( KC )
      RUVPR4 = 0.5 * TEMPR * POR( 4 )
C
      TEMPL = ( 1 + ISNL ) * PPR( KC ) +
     .              ( 1 - ISNL ) * PPL( KC )
      RUVPL4 = 0.5 * TEMPL * POL( 4 )
C
      PMIN( KC ) = AMIN1( 1. , RUVPR1 , RUVPL1 , RUVPR2 , RUVPL2 ,
     .                         RUVPR3 , RUVPL3 , RUVPR4 , RUVPL4 )
C
  170 CONTINUE
C
      DO 330 IH = 1 , 3
C
      DO 330 IC = NC1 , NC2
         KC = IC - NC1 + 1
```

```
C
        RGRAD( IC , IH ) = RGRAD( IC , IH ) * RMIN( KC )
        UGRAD( IC , IH ) = UGRAD( IC , IH ) * UMIN( KC )
        VGRAD( IC , IH ) = VGRAD( IC , IH ) * VMIN( KC )
        WGRAD( IC , IH ) = WGRAD( IC , IH ) * WMIN( KC )
        PGRAD( IC , IH ) = PGRAD( IC , IH ) * PMIN( KC )
C
  330   CONTINUE
C
       NC1 = NC2 + 1
       NC2 = NC2 + NOFVEC( INC + 1 )
 80     CONTINUE
C
       CALL FCHART
C
       RETURN
       END
C
```

Thu Jul 1 14:17:00 1993 threed.f SUBROUTINE FIRST

```
       SUBROUTINE FIRST
C
C-----------------------------------------------------------------I
C                                                                 I
C       FIRST IS TO BY PASS GRADIENT AND CHARACTERSTIC COMPUTATION I
C                                                                 I
C-----------------------------------------------------------------I
C
       include     'dmsh00.h'
       include     'dhydm0.h'
       include     'dphsm0.h'
       include     'dmtrl0.h'
C
       DO 110 IS = 1 , NS
C
        ICL = JS( 7 , IS )
        ICR = JS( 8 , IS )
C
        RL( IS ) = HYDV( ICL , 1 )
        UL( IS ) = HYDV( ICL , 2 ) * XN( IS ) +
     .             HYDV( ICL , 3 ) * YN( IS ) +
     .             HYDV( ICL , 4 ) * ZN( IS )
        VL( IS ) = HYDV( ICL , 2 ) * XP( IS ) +
     .             HYDV( ICL , 3 ) * YP( IS ) +
     .             HYDV( ICL , 4 ) * ZP( IS )
        WL( IS ) = HYDV( ICL , 2 ) * XT( IS ) +
     .             HYDV( ICL , 3 ) * YT( IS ) +
     .             HYDV( ICL , 4 ) * ZT( IS )
        PL( IS ) = HYDV( ICL , 5 )
        AL( IS ) = HYDV( ICL , 6 )
        GL( IS ) = HYDV( ICL , 7 )
        EL( IS ) = HYDV( ICL , 8 )
C
       IATRB = JS( 9 , IS )
       IF( IATRB . EQ . 0 ) THEN
C
        RR( IS ) = HYDV( ICR , 1 )
        UR( IS ) = HYDV( ICR , 2 ) * XN( IS ) +
     .             HYDV( ICR , 3 ) * YN( IS ) +
     .             HYDV( ICR , 4 ) * ZN( IS )
        VR( IS ) = HYDV( ICR , 2 ) * XP( IS ) +
     .             HYDV( ICR , 3 ) * YP( IS ) +
     .             HYDV( ICR , 4 ) * ZP( IS )
        WR( IS ) = HYDV( ICR , 2 ) * XT( IS ) +
     .             HYDV( ICR , 3 ) * YT( IS ) +
     .             HYDV( ICR , 4 ) * ZT( IS )
        PR( IS ) = HYDV( ICR , 5 )
        AR( IS ) = HYDV( ICR , 6 )
        GR( IS ) = HYDV( ICR , 7 )
        ER( IS ) = HYDV( ICR , 8 )
C
       ELSE
C
```

```
      IF( IATRB . EQ . 8 ) THEN
C
       RR( IS ) = RIN
       UR( IS ) = UIN * XN( IS ) + VIN * YN( IS ) + WIN * ZN( IS )
       VR( IS ) = UIN * XP( IS ) + VIN * YP( IS ) + WIN * ZP( IS )
       WR( IS ) = UIN * XT( IS ) + VIN * YT( IS ) + WIN * ZT( IS )
       PR( IS ) = PIN
       AR( IS ) = AL( IS )
       GR( IS ) = GL( IS )
       ER( IS ) = EL( IS )
C
      END IF
C
      IF( IATRB . EQ . 7 ) THEN
C
         RR( IS ) = RL( IS )
         UR( IS ) = UL( IS )
         VR( IS ) = VL( IS )
         WR( IS ) = WL( IS )
         PR( IS ) = PL( IS )
         AR( IS ) = AL( IS )
         GR( IS ) = GL( IS )
         ER( IS ) = EL( IS )
C
      END IF
C
      IF( IATRB . EQ . 6 ) THEN
C
         RR( IS ) = RL( IS )
         UR( IS ) = - UL( IS )
         VR( IS ) = VL( IS )
         WR( IS ) = WL( IS )
         PR( IS ) = PL( IS )
         AR( IS ) = AL( IS )
         GR( IS ) = GL( IS )
         ER( IS ) = EL( IS )
C
      END IF
C
      END IF
C
  110 CONTINUE
C
      RETURN
      END
C
```

```
      SUBROUTINE FCHART
C
C-------------------------------------------------------------------I
C                                                                   I
C      CHARCT INTRODUCE CORRECTION FOR SECOND ORDER CALCULATION     I
C                                                                   I
C-------------------------------------------------------------------I
C
      include     'dmsh00.h'
      include     'dhydm0.h'
      include     'dphsm0.h'
      include     'dmtrl0.h'
C
      REAL ZZLEFT(128),ZOLEFT(128),ZPLEFT(128),ZMLEFT(128)
      REAL ZZRIGT(128),ZORIGT(128),ZPRIGT(128),ZMRIGT(128)
      REAL UPLEFT(128),UMLEFT(128),URLEFT(128)
      REAL UPRIGT(128),UMRIGT(128),URRIGT(128)
      REAL UVLEFT(128),UVRIGT(128),CNLEFT(128),CNRIGT(128)
      REAL RLEFTT(128),ULEFTT(128),VLEFTT(128),PLEFTT(128),
     .     ALEFTT(128)
      REAL RRIGHT(128),URIGHT(128),VRIGHT(128),PRIGHT(128),
     .     ARIGHT(128)
C
      NS1 = 1
      NS2 = NOFVES( 1 )
      DO 90 INS = 1 , NVEES
C
      DO 110 IS = NS1 , NS2
         KS = IS - NS1 + 1
C
       ICL = JS( 7 , IS )
       ICR = JS( 8 , IS )
C
       GL( IS ) = HYDV( ICL , 7 )
       CNLFTS = GL( IS )  * HYDV( ICL , 5 ) / HYDV( ICL , 1 )
       CNLFT = SQRT( CNLFTS )
C
      IATRB = JS( 9 , IS )
      IF( IATRB . EQ . 0 ) THEN
C
       XYZ = 1. / XS( 5 , IS )
         XXN = ( XC( 1 , ICR ) - XC( 1 , ICL ) ) * XYZ
         YYN = ( XC( 2 , ICR ) - XC( 2 , ICL ) ) * XYZ
         ZZN = ( XC( 3 , ICR ) - XC( 3 , ICL ) ) * XYZ
C
         UVLFT = HYDV( ICL , 2 ) * XXN +
     .           HYDV( ICL , 3 ) * YYN +
     .           HYDV( ICL , 4 ) * ZZN
C
       GR( IS ) = HYDV( ICR , 7 )
       CNRGTS = GR( IS ) * HYDV( ICR , 5 ) / HYDV( ICR , 1 )
       CNRGT = SQRT( CNRGTS )
C
         UVRGT = HYDV( ICR , 2 ) * XXN +
     .           HYDV( ICR , 3 ) * YYN +
     .           HYDV( ICR , 4 ) * ZZN
C
       ELSE
C
       CNRGT = CNLFT
C
       XYZ = 1. / XS( 5 , IS )
         XXN = ( XYZMDL( 1 , IS ) - XC( 1 , ICL ) ) * XYZ
         YYN = ( XYZMDL( 2 , IS ) - XC( 2 , ICL ) ) * XYZ
         ZZN = ( XYZMDL( 3 , IS ) - XC( 3 , ICL ) ) * XYZ
C
         UVLFT = HYDV( ICL , 2 ) * XXN +
     .           HYDV( ICL , 3 ) * YYN +
     .           HYDV( ICL , 4 ) * ZZN
C
         UVRGT = UVLFT
         GR( IS ) = GL( IS )
C
       END IF
```

```
C
            CNLEFT( KS ) = CNLFT
            CNRIGT( KS ) = CNRGT
C
            UVLEFT( KS ) = UVLFT
            UVRIGT( KS ) = UVRGT
C
  110 CONTINUE
C
        DO 130 KS = 1 , NOFVES( INS )
C
        ZZLEFT( KS ) =   .5 * ( UVLEFT( KS ) + CNLEFT( KS ) ) * DTT
        ZZRIGT( KS ) = - .5 * ( UVRIGT( KS ) - CNRIGT( KS ) ) * DTT
C
  130 CONTINUE
C
C  CHARACTERISTICS LOCATIONS
C
        DO 140 KS = 1 , NOFVES( INS )
C
        IF( ZZLEFT( KS ) . LT . 0. ) ZZLEFT( KS ) = 0.
        IF( ZZRIGT( KS ) . LT . 0. ) ZZRIGT( KS ) = 0.
C
  140 CONTINUE
C
        DO 150 KS = 1 , NOFVES( INS )
C
        ZOLEFT( KS ) =   .5 * UVLEFT( KS ) * DTT
        ZORIGT( KS ) = - .5 * UVRIGT( KS ) * DTT
        ZPLEFT( KS ) =   .5 * ( UVLEFT( KS ) + CNLEFT( KS ) ) * DTT
        ZPRIGT( KS ) = - .5 * ( UVRIGT( KS ) + CNRIGT( KS ) ) * DTT
        ZMLEFT( KS ) =   .5 * ( UVLEFT( KS ) - CNLEFT( KS ) ) * DTT
        ZMRIGT( KS ) = - .5 * ( UVRIGT( KS ) - CNRIGT( KS ) ) * DTT
C
  150 CONTINUE
C
C  FIRST GUESS LEFT AND RIGHT VARIABLES, LINEAR INTERPOLATON
C
        DO 160 IS = NS1 , NS2
           KS = IS - NS1 + 1
C
         ICL = JS( 7 , IS )
         ICR = JS( 8 , IS )
C
        IATRB = JS( 9 , IS )
        IF( IATRB . EQ . 0 ) THEN
C
         XYZ = 1. / XS( 5 , IS )
          XXN = ( XC( 1 , ICR ) - XC( 1 , ICL ) ) * XYZ
          YYN = ( XC( 2 , ICR ) - XC( 2 , ICL ) ) * XYZ
          ZZN = ( XC( 3 , ICR ) - XC( 3 , ICL ) ) * XYZ
C
          XXL = ( XYZMDL( 1 , IS ) - XC( 1 , ICL ) )
          YYL = ( XYZMDL( 2 , IS ) - XC( 2 , ICL ) )
          ZZL = ( XYZMDL( 3 , IS ) - XC( 3 , ICL ) )
C
        XX = XXL - ZZLEFT( KS ) * XXN
        YY = YYL - ZZLEFT( KS ) * YYN
        ZZ = ZZL - ZZLEFT( KS ) * ZZN
C
         HRRL =  HYDV( ICL , 1 ) + RGRAD( ICL , 1 ) * XX +
     .    RGRAD( ICL , 2 ) * YY + RGRAD( ICL , 3 ) * ZZ
         HUUL =  HYDV( ICL , 2 ) + UGRAD( ICL , 1 ) * XX +
     .    UGRAD( ICL , 2 ) * YY + UGRAD( ICL , 3 ) * ZZ
         HVVL =  HYDV( ICL , 3 ) + VGRAD( ICL , 1 ) * XX +
     .    VGRAD( ICL , 2 ) * YY + VGRAD( ICL , 3 ) * ZZ
         HWWL =  HYDV( ICL , 4 ) + WGRAD( ICL , 1 ) * XX +
     .    WGRAD( ICL , 2 ) * YY + WGRAD( ICL , 3 ) * ZZ
         HPPL =  HYDV( ICL , 5 ) + PGRAD( ICL , 1 ) * XX +
     .    PGRAD( ICL , 2 ) * YY + PGRAD( ICL , 3 ) * ZZ
C
         GMTLFT = GL( IS ) * HRRL * HPPL
         SQGMTL = SQRT( GMTLFT )
C
```

```
      XX = ( ZPLEFT( KS ) - ZZLEFT( KS ) ) * XXN
      YY = ( ZPLEFT( KS ) - ZZLEFT( KS ) ) * YYN
      ZZ = ( ZPLEFT( KS ) - ZZLEFT( KS ) ) * ZZN
      UUU = UGRAD( ICL , 1 ) * XX + UGRAD( ICL , 2 ) * YY +
     .      UGRAD( ICL , 3 ) * ZZ
      PPP = PGRAD( ICL , 1 ) * XX + PGRAD( ICL , 2 ) * YY +
     .      PGRAD( ICL , 3 ) * ZZ
      UPLFT = - .5 * ( UUU + PPP / SQGMTL ) / SQGMTL
C
      XX = ( ZMLEFT( KS ) - ZZLEFT( KS ) ) * XXN
      YY = ( ZMLEFT( KS ) - ZZLEFT( KS ) ) * YYN
      ZZ = ( ZMLEFT( KS ) - ZZLEFT( KS ) ) * ZZN
      UUU = UGRAD( ICL , 1 ) * XX + UGRAD( ICL , 2 ) * YY +
     .      UGRAD( ICL , 3 ) * ZZ
      PPP = PGRAD( ICL , 1 ) * XX + PGRAD( ICL , 2 ) * YY +
     .      PGRAD( ICL , 3 ) * ZZ
      UMLFT =   .5 * ( UUU - PPP / SQGMTL ) / SQGMTL
C
      XX = ( ZOLEFT( KS ) - ZZLEFT( KS ) ) * XXN
      YY = ( ZOLEFT( KS ) - ZZLEFT( KS ) ) * YYN
      ZZ = ( ZOLEFT( KS ) - ZZLEFT( KS ) ) * ZZN
      PPP = PGRAD( ICL , 1 ) * XX + PGRAD( ICL , 2 ) * YY +
     .      PGRAD( ICL , 3 ) * ZZ
C
      XX = XXL - ZOLEFT( KS ) * XXN
      YY = YYL - ZOLEFT( KS ) * YYN
      ZZ = ZZL - ZOLEFT( KS ) * ZZN
C
      RRRR = HYDV( ICL , 1 ) + RGRAD( ICL , 1 ) * XX +
     .                    RGRAD( ICL , 2 ) * YY + RGRAD( ICL , 3 ) * ZZ
      URLFT = PPP / GMTLFT + 1. / HRRL - 1. / RRRR
C
         XXR = ( XYZMDL( 1 , IS ) - XC( 1 , ICR ) )
         YYR = ( XYZMDL( 2 , IS ) - XC( 2 , ICR ) )
         ZZR = ( XYZMDL( 3 , IS ) - XC( 3 , ICR ) )
C
      XX = XXR + ZZRIGT( KS ) * XXN
      YY = YYR + ZZRIGT( KS ) * YYN
      ZZ = ZZR + ZZRIGT( KS ) * ZZN
C
       HRRR =  HYDV( ICR , 1 ) + RGRAD( ICR , 1 ) * XX +
     .   RGRAD( ICR , 2 ) * YY + RGRAD( ICR , 3 ) * ZZ
       HUUR =  HYDV( ICR , 2 ) + UGRAD( ICR , 1 ) * XX +
     .   UGRAD( ICR , 2 ) * YY + UGRAD( ICR , 3 ) * ZZ
       HVVR =  HYDV( ICR , 3 ) + VGRAD( ICR , 1 ) * XX +
     .   VGRAD( ICR , 2 ) * YY + VGRAD( ICR , 3 ) * ZZ
       HWWR =  HYDV( ICR , 4 ) + WGRAD( ICR , 1 ) * XX +
     .   WGRAD( ICR , 2 ) * YY + WGRAD( ICR , 3 ) * ZZ
       HPPR =  HYDV( ICR , 5 ) + PGRAD( ICR , 1 ) * XX +
     .   PGRAD( ICR , 2 ) * YY + PGRAD( ICR , 3 ) * ZZ
C
       GMTRGT = GR( IS ) * HRRR * HPPR
       SQGMTR = SQRT( GMTRGT )
C
      XX = ( ZZRIGT( KS ) - ZPRIGT( KS ) ) * XXN
      YY = ( ZZRIGT( KS ) - ZPRIGT( KS ) ) * YYN
      ZZ = ( ZZRIGT( KS ) - ZPRIGT( KS ) ) * ZZN
      UUU = UGRAD( ICR , 1 ) * XX + UGRAD( ICR , 2 ) * YY +
     .      UGRAD( ICR , 3 ) * ZZ
      PPP = PGRAD( ICR , 1 ) * XX + PGRAD( ICR , 2 ) * YY +
     .      PGRAD( ICR , 3 ) * ZZ
      UPRGT = - .5 * ( UUU + PPP / SQGMTR ) / SQGMTR
C
      XX = ( ZZRIGT( KS ) - ZMRIGT( KS ) ) * XXN
      YY = ( ZZRIGT( KS ) - ZMRIGT( KS ) ) * YYN
      ZZ = ( ZZRIGT( KS ) - ZMRIGT( KS ) ) * ZZN
      UUU = UGRAD( ICR , 1 ) * XX + UGRAD( ICR , 2 ) * YY +
     .      UGRAD( ICR , 3 ) * ZZ
      PPP = PGRAD( ICR , 1 ) * XX + PGRAD( ICR , 2 ) * YY +
     .      PGRAD( ICR , 3 ) * ZZ
      UMRGT =   .5 * ( UUU - PPP / SQGMTR ) / SQGMTR
C
      XX = ( ZZRIGT( KS ) - ZORIGT( KS ) ) * XXN
      YY = ( ZZRIGT( KS ) - ZORIGT( KS ) ) * YYN
```

```
      ZZ = ( ZZRIGT( KS ) - ZORIGT( KS ) ) * ZZN
      PPP = PGRAD( ICR , 1 ) * XX + PGRAD( ICR , 2 ) * YY +
     .      PGRAD( ICR , 3 ) * ZZ
C
      XX = XXR + ZORIGT( KS ) * XXN
      YY = YYR + ZORIGT( KS ) * YYN
      ZZ = ZZR + ZORIGT( KS ) * ZZN
C
      RRRR = HYDV( ICR , 1 ) + RGRAD( ICR , 1 ) * XX +
     .                    RGRAD( ICR , 2 ) * YY + RGRAD( ICR , 3 ) * ZZ
      URRGT = PPP / GMTRGT + 1. / HRRR - 1. / RRRR
C
      ELSE
C
       XYZ = 1. / XS( 5 , IS )
         XXN = ( XYZMDL( 1 , IS ) - XC( 1 , ICL ) ) * XYZ
         YYN = ( XYZMDL( 2 , IS ) - XC( 2 , ICL ) ) * XYZ
         ZZN = ( XYZMDL( 3 , IS ) - XC( 3 , ICL ) ) * XYZ
C
         XXL = ( XYZMDL( 1 , IS ) - XC( 1 , ICL ) )
         YYL = ( XYZMDL( 2 , IS ) - XC( 2 , ICL ) )
         ZZL = ( XYZMDL( 3 , IS ) - XC( 3 , ICL ) )
C
      XX = XXL - ZZLEFT( KS ) * XXN
      YY = YYL - ZZLEFT( KS ) * YYN
      ZZ = ZZL - ZZLEFT( KS ) * ZZN
C
       HRRL =  HYDV( ICL , 1 ) + RGRAD( ICL , 1 ) * XX +
     .   RGRAD( ICL , 2 ) * YY + RGRAD( ICL , 3 ) * ZZ
       HUUL =  HYDV( ICL , 2 ) + UGRAD( ICL , 1 ) * XX +
     .   UGRAD( ICL , 2 ) * YY + UGRAD( ICL , 3 ) * ZZ
       HVVL =  HYDV( ICL , 3 ) + VGRAD( ICL , 1 ) * XX +
     .   VGRAD( ICL , 2 ) * YY + VGRAD( ICL , 3 ) * ZZ
       HWWL =  HYDV( ICL , 4 ) + WGRAD( ICL , 1 ) * XX +
     .   WGRAD( ICL , 2 ) * YY + WGRAD( ICL , 3 ) * ZZ
       HPPL =  HYDV( ICL , 5 ) + PGRAD( ICL , 1 ) * XX +
     .   PGRAD( ICL , 2 ) * YY + PGRAD( ICL , 3 ) * ZZ
C
       GMTLFT = GL( IS ) * HRRL * HPPL
       SQGMTL = SQRT( GMTLFT )
C
      XX = ( ZPLEFT( KS ) - ZZLEFT( KS ) ) * XXN
      YY = ( ZPLEFT( KS ) - ZZLEFT( KS ) ) * YYN
      ZZ = ( ZPLEFT( KS ) - ZZLEFT( KS ) ) * ZZN
      UUU = UGRAD( ICL , 1 ) * XX + UGRAD( ICL , 2 ) * YY +
     .      UGRAD( ICL , 3 ) * ZZ
      PPP = PGRAD( ICL , 1 ) * XX + PGRAD( ICL , 2 ) * YY +
     .      PGRAD( ICL , 3 ) * ZZ
      UPLFT = - .5 * ( UUU + PPP / SQGMTL ) / SQGMTL
C
      XX = ( ZMLEFT( KS ) - ZZLEFT( KS ) ) * XXN
      YY = ( ZMLEFT( KS ) - ZZLEFT( KS ) ) * YYN
      ZZ = ( ZMLEFT( KS ) - ZZLEFT( KS ) ) * ZZN
      UUU = UGRAD( ICL , 1 ) * XX + UGRAD( ICL , 2 ) * YY +
     .      UGRAD( ICL , 3 ) * ZZ
      PPP = PGRAD( ICL , 1 ) * XX + PGRAD( ICL , 2 ) * YY +
     .      PGRAD( ICL , 3 ) * ZZ
      UMLFT =   .5 * ( UUU - PPP / SQGMTL ) / SQGMTL
C
      XX = ( ZOLEFT( KS ) - ZZLEFT( KS ) ) * XXN
      YY = ( ZOLEFT( KS ) - ZZLEFT( KS ) ) * YYN
      ZZ = ( ZOLEFT( KS ) - ZZLEFT( KS ) ) * ZZN
      PPP = PGRAD( ICL , 1 ) * XX + PGRAD( ICL , 2 ) * YY +
     .      PGRAD( ICL , 3 ) * ZZ
C
      XX = XXL - ZOLEFT( KS ) * XXN
      YY = YYL - ZOLEFT( KS ) * YYN
      ZZ = ZZL - ZOLEFT( KS ) * ZZN
C
      RRRR = HYDV( ICL , 1 ) + RGRAD( ICL , 1 ) * XX +
     .                    RGRAD( ICL , 2 ) * YY + RGRAD( ICL , 3 ) * ZZ
      URLFT = PPP / GMTLFT + 1. / HRRL - 1. / RRRR
C
       HRRR =  HRRL
```

```
            HUUR =  HUUL
            HVVR =  HVVL
            HWWR =  HWWL
            HPPR =  HPPL
C
            GMTRGT = GMTLFT
            SQGMTR = SQGMTL
C
           UPRGT = UPLFT
           UMRGT = UMLFT
           URRGT = URLFT
C
           END IF
C
           RRL( KS ) = HRRL
           UUL( KS ) = HUUL
           VVL( KS ) = HVVL
           WWL( KS ) = HWWL
           PPL( KS ) = HPPL
C
           RRR( KS ) = HRRR
           UUR( KS ) = HUUR
           VVR( KS ) = HVVR
           WWR( KS ) = HWWR
           PPR( KS ) = HPPR
C
           UPLEFT( KS ) = UPLFT
           UMLEFT( KS ) = UMLFT
           URLEFT( KS ) = URLFT
C
           UPRIGT( KS ) = UPRGT
           UMRIGT( KS ) = UMRGT
           URRIGT( KS ) = URRGT
C
 160  CONTINUE
C
C  CORRECTION OF THE FIRST GUESS
C
           DO 170 KS = 1 , NOFVES( INS )
C
           IF( UVLEFT( KS ) + CNLEFT( KS ) . LE . 0. ) UPLEFT( KS ) = 0.
           IF( UVLEFT( KS ) - CNLEFT( KS ) . LE . 0. ) UMLEFT( KS ) = 0.
           IF( UVLEFT( KS ) .                 LE . 0. ) URLEFT( KS ) = 0.
C
           IF( UVRIGT( KS ) + CNRIGT( KS ) . GE . 0. ) UPRIGT( KS ) = 0.
           IF( UVRIGT( KS ) - CNRIGT( KS ) . GE . 0. ) UMRIGT( KS ) = 0.
           IF( UVRIGT( KS ) .                 GE . 0. ) URRIGT( KS ) = 0.
C
 170  CONTINUE
C
C  FINAL VALUES FOR RIGHT AND LEFT STATES
C
           DO 180 KS = 1 , NOFVES( INS )
               IS = KS + NS1 - 1
C
            GMTLFT = GL( IS ) * RRL( KS ) * PPL( KS )
            SQGMTL = SQRT( GMTLFT )
C
            GMTRGT = GR( IS ) * RRR( KS ) * PPR( KS )
            SQGMTR = SQRT( GMTRGT )
C
            RRL( KS ) = 1. / ( 1. / RRL( KS ) - ( UPLEFT( KS ) +
     .                           UMLEFT( KS ) + URLEFT( KS ) ) )
            UUL( KS ) = UUL( KS ) + SQGMTL * ( UPLEFT( KS ) -
     .                                         UMLEFT( KS ) )
            VVL( KS ) = VVL( KS ) + SQGMTL * ( UPLEFT( KS ) -
     .                                         UMLEFT( KS ) )
            WWL( KS ) = WWL( KS ) + SQGMTL * ( UPLEFT( KS ) -
     .                                         UMLEFT( KS ) )
            PPL( KS ) = PPL( KS ) + GMTLFT * ( UPLEFT( KS ) +
     .                                         UMLEFT( KS ) )
C
            RRR( KS ) = 1. / ( 1. / RRR( KS ) - ( UPRIGT( KS ) +
     .                           UMRIGT( KS ) + URRIGT( KS ) ) )
```

```
      UUR( KS ) = UUR( KS ) + SQGMTR * ( UPRIGT( KS ) -
     .                                    UMRIGT( KS ) )
      VVR( KS ) = VVR( KS ) + SQGMTR * ( UPRIGT( KS ) -
     .                                    UMRIGT( KS ) )
      WWR( KS ) = WWR( KS ) + SQGMTR * ( UPRIGT( KS ) -
     .                                    UMRIGT( KS ) )
      PPR( KS ) = PPR( KS ) + GMTRGT * ( UPRIGT( KS ) +
     .                                    UMRIGT( KS ) )
C
  180 CONTINUE
C
      DO 200 IS = NS1 , NS2
         KS = IS - NS1 + 1
C
       ICL = JS( 7 , IS )
       ICR = JS( 8 , IS )
C
       RL( IS ) = RRL( KS )
       UL( IS ) = UUL( KS ) * XN( IS ) +
     .            VVL( KS ) * YN( IS ) +
     .            WWL( KS ) * ZN( IS )
       VL( IS ) = UUL( KS ) * XP( IS ) +
     .            VVL( KS ) * YP( IS ) +
     .            WWL( KS ) * ZP( IS )
       WL( IS ) = UUL( KS ) * XT( IS ) +
     .            VVL( KS ) * YT( IS ) +
     .            WWL( KS ) * ZT( IS )
       PL( IS ) = PPL( KS )
       AL( IS ) = HYDV( ICL , 6 )
       GL( IS ) = HYDV( ICL , 7 )
       EL( IS ) = HYDV( ICL , 8 )
C
      IATRB = JS( 9 , IS )
      IF( IATRB . EQ . 0 ) THEN
C
       RR( IS ) = RRR( KS )
       UR( IS ) = UUR( KS ) * XN( IS ) +
     .            VVR( KS ) * YN( IS ) +
     .            WWR( KS ) * ZN( IS )
       VR( IS ) = UUR( KS ) * XP( IS ) +
     .            VVR( KS ) * YP( IS ) +
     .            WWR( KS ) * ZP( IS )
       WR( IS ) = UUR( KS ) * XT( IS ) +
     .            VVR( KS ) * YT( IS ) +
     .            WWR( KS ) * ZT( IS )
       PR( IS ) = PPR( KS )
       AR( IS ) = HYDV( ICR , 6 )
       GR( IS ) = HYDV( ICR , 7 )
       ER( IS ) = HYDV( ICR , 8 )
C
      ELSE
C
      IF( IATRB . EQ . 8 ) THEN
C
       RR( IS ) = RIN
       UR( IS ) = UIN * XN( IS ) + VIN * YN( IS ) + WIN * ZN( IS )
       VR( IS ) = UIN * XP( IS ) + VIN * YP( IS ) + WIN * ZP( IS )
       WR( IS ) = UIN * XT( IS ) + VIN * YT( IS ) + WIN * ZT( IS )
       PR( IS ) = PIN
       AR( IS ) = AL( IS )
       GR( IS ) = GL( IS )
       ER( IS ) = EL( IS )
C
      END IF
C
      IF( IATRB . EQ . 7 ) THEN
C
         RR( IS ) = RL( IS )
         UR( IS ) = UL( IS )
         VR( IS ) = VL( IS )
         WR( IS ) = WL( IS )
         PR( IS ) = PL( IS )
         AR( IS ) = AL( IS )
         GR( IS ) = GL( IS )
```

```
            ER( IS ) = EL( IS )
C
         END IF
C
         IF( IATRB . EQ . 6 ) THEN
C
            RR( IS ) = RL( IS )
            UR( IS ) = - UL( IS )
            VR( IS ) = VL( IS )
            WR( IS ) = WL( IS )
            PR( IS ) = PL( IS )
            AR( IS ) = AL( IS )
            GR( IS ) = GL( IS )
            ER( IS ) = EL( IS )
C
         END IF
C
         END IF
C
 200     CONTINUE
C
         NS1 = NS2 + 1
         NS2 = NS2 + NOFVES( INS + 1 )
  90     CONTINUE
C
         RETURN
         END
```

Thu Jul 1 14:17:00 1993 threed.f SUBROUTINE EOS1

```
         SUBROUTINE EOS1 (RRR,EEE,N,GAMMA)
C
C        AIR IS ASSUMED TO BE CALORICALLY IMPERFECT, THERMALLY PERFECT.
C        THEREFORE, INCLUDE IMPERFECTIONS VIA A VARIABLE GAMMA DEPENDENT
C        ON DENSITY AND INTERNAL ENERGY.  THIS ROUTINE PERFORMS A TABLE
C        LOOK UP FOR GAMMA.
C
C        INPUT VARIBLE DEFINITIONS.
C        RRR = MASS DENSITY
C        EEE = INTERNAL ENERGY PER UNIT VOLUME
C              (CONVERTED FOR INTERNAL *CALL TO ENERGY PER UNIT MASS)
C        N   = NUMBER OF ENTRIES IN ARRAYS RRR & EEE
C
         PARAMETER (M = 64)
C
         DIMENSION RRR(N), EEE(N), GAMMA(N)
         DIMENSION T11(M), T12(M), T21(M), T22(M), RHO(M), E(M)
         DIMENSION OMP(M), Q(M), I(M), J(M)
         DIMENSION G1(168),G2(112),G3(112),G4(112),G5(112),
     !             G6(112),G7(112),GF(840)
C
C        NOTE:  THE TABLE LOOK UP TREATS ARRAY GF AS THOUGH IT
C        WERE DIMENSIONED (8,105).
C
         EQUIVALENCE (G1(1),GF(  1)), (G2(1),GF(169)), (G3(1),GF(281)),
     !               (G4(1),GF(393)), (G5(1),GF(505)), (G6(1),GF(617)),
     !               (G7(1),GF(729))
C
         DATA XL16E  /2.772588722239774483568908181041479110717773437 5/
C------------------------------------------------------------- --
C        G = GAMMA - 1.0 IS STORED FOR 32 BIT WORD MACHINES IN POWERS OF
C        16 ACROSS FOR MASS DENSITY VARIATION AND INTERMEDIATE VALUES
C        1 - 16 FOR POWERS OF 16 VERTICALLY WHICH REPRESENT THE INTERNAL
C        ENERGY VARIATION.
C
C        16**(?)  .GE. RHO .GE. 16**(-6)
C        16**(15) .GE.  E  .GE. 16**(8)
C---------------------------------------------------------------
         DATA G1 /8*.4222,8*.4152,8*.4110,8*.4081,8*.4058,8*.4040,
     !            8*.4024,8*.4011,8*.3998,8*.3988,8*.3978,8*.3969,
     !            8*.3961,8*.3953,8*.3935,8*.3918,
     !           .3723,.3715,.3707,.3699,.3690,.3680,.3663,.3637,
     !           .3555,.3538,.3522,.3502,.3476,.3430,.3344,.3238,
     !           .3370,.3370,.3370,.3364,.3347,.3277,.3099,.2885,
```

```
     !          .3257,.3227,.3201,.3134,.3062,.3014,.2884,.2591,
     !          .3166,.3110,.3063,.2946,.2831,.2783,.2677,.2358/
      DATA G2/.[illegible],.3006,.2940,.2787,.2635,.2588,.2502,.2236,
     !          .3075,.2906,.2810,.2665,.2466,.2418,.2350,.2131,
     !          .3043,.2819,.2695,.2554,.2317,.2269,.2216,.2038,
     !          .2929,.2740,.2593,.2455,.2206,.2136,.2097,.1955,
     !          .2840,.2672,.2500,.2366,.2166,.2015,.1988,.1879,
     !          .2764,.2611,.2429,.2285,.2125,.1890,.1890,.1811,
     !          .2714,.2555,.[illegible],.2210,.2079,.1818,.1799,.1747,
     !          .2669,.[illegible],.2343,.2141,.2037,.1822,.1709,.1689,
     !          .2624,.2473,.2304,.2096,.1998,.1828,.1684,.1639,
     !          .2595,.2446,.2268,.2087,.1961,.1834,.1673,.1601,
     !          .2401,.2191,.1972,.1775,.1592,.1444,.1358,.1203,
     !          .2002,.1960,.1749,.1536,.1376,.1252,.1107,.1044,
     !          .1911,.1829,.1633,.1420,.1266,.1101,.1012,.0933,
     !          .1950,.1781,.1566,.1415,.1241,.1118,.1009,.0948/
      DATA G3/.2001,.1789,.1594,.1443,.1306,.1189,.1095,.1013,
     !          .2040,.1826,.1657,.1494,.1338,.1177,.1081,.0980,
     !          .2034,.1854,.1683,.1497,.1322,.1169,.1051,.0946,
     !          .1969,.1855,.1685,.1487,.1304,.1149,.1024,.0916,
     !          .1899,.1837,.1677,.1475,.1287,.1126,.1002,.0900,
     !          .1841,.1817,.1667,.1464,.1272,.1109,.0983,.0888,
     !          .1800,.1800,.1659,.1455,.1262,.1097,.0965,.0878,
     !          .1779,.1787,.1657,.1450,.1254,.1087,.0949,.0868,
     !          .1773,.1778,.1656,.1447,.1250,.1080,.0939,.0859,
     !          .1783,.1778,.1658,.1448,.1248,.1076,.0933,.0851,
     !          .1808,.1781,.1667,.1451,.1248,.1074,.0930,.0843,
     !          .2134,.2040,.1978,.1782,.1565,.1368,.1206,.1074,
     !          .2210,.2072,.1957,.1739,.1516,.1312,.1137,.1000,
     !          .2245,.2109,.1989,.1772,.1563,.1390,.1247,.1133/
      DATA G4/.2299,.2132,.2017,.1795,.1579,.1384,.1221,.1090,
     !          .2350,.2157,.2023,.1798,.1575,.1370,.1197,.1057,
     !          .2397,.2194,.2034,.1796,.1572,.1372,.1205,.1070,
     !          .2452,.2227,.2050,.1805,.1576,.1379,.1236,.1118,
     !          .2510,.2256,.2069,.1814,.1581,.1383,.1231,.1103,
     !          .2560,.2282,.2091,.1822,.1585,.1385,.1226,.1083,
     !          .2605,.2312,.2111,.1829,.1588,.1386,.1222,.1070,
     !          .2677,.2358,.2129,.1836,.1592,.1386,.1218,.1071,
     !          .2759,.2403,.2145,.1857,.1598,.1389,.1219,.1078,
     !          .2834,.2445,.2160,.1878,.1603,.1394,.1223,.1084,
     !          .2905,.2484,.2175,.1898,.1613,.1399,.1226,.1090,
     !          .2963,.2531,.2199,.1918,.1625,.1407,.1230,.1096,
     !          .4323,.3582,.3109,.2889,.2803,.2706,.2410,.2224,
     !          .4610,.4026,.3624,.3212,.2926,.2551,.2375,.2015/
      DATA G5/.4199,.3837,.3401,.2979,.2623,.2318,.2108,.1854,
     !          .3924,.3642,.3194,.2760,.2427,.2157,.1902,.1721,
     !          .3794,.3479,.3025,.2673,.2311,.2019,.1842,.1613,
     !          .3674,.3448,.2961,.2593,.2255,.1994,.1785,.1594,
     !          .3573,.3443,.2910,.2517,.2293,.2006,.1843,.1679,
     !          .3661,.3438,.2935,.2597,.2336,.2225,.2143,.2116,
     !          .3674,.3435,.3080,.2728,.2606,.2577,.2573,.2573,
     !          .3685,.3453,.3210,.3014,.2942,.2933,.2932,.2932,
     !          .3814,.3612,.3341,.3276,.3257,.3253,.3252,.3252,
     !          .3903,.3752,.3570,.3522,.3513,.3510,.3506,.3496,
     !          .4012,.3899,.3782,.3751,.3743,.3741,.3734,.3713,
     !          .4155,.4057,.3956,.3930,.3920,.3913,.3907,.3890,
     !          .4290,.4205,.4118,.4092,.4077,.4065,.4059,.4047,
     !          .5411,.5385,.5359,.5353,.5351,.5350,.5350,.5350/
      DATA G6/.5823,.5812,.5801,.5797,.5796,.5797,.5797,.5797,
     !          .6096,.6090,.6085,.6082,.6082,.6083,.6083,.6083,
     !          .6308,.6306,.6305,.6303,.6303,.6305,.6305,.6305,
     !          .6481,.6483,.6485,.6483,.6484,.6486,.6487,.6487,
     !          .6627,.6632,.6637,.6636,.6637,.6640,.6640,.6640,
     !          .6754,.6761,.6769,.6768,.6770,.6773,.6773,.6773,
     !          .6866,.6875,.6885,.6884,.6886,.6890,.6890,.6890,
     !          .6966,.6977,.6989,.6989,.6991,.6995,.6995,.6995,
     !          .7056,.7070,.7083,.7083,.7085,.7090,.7090,.7090,
     !          .7139,.7154,.7169,.7169,.7172,.7176,.7177,.7177,
     !          .7214,.7231,.7248,.7248,.7251,.7256,.7256,.7256,
     !          .7285,.7303,.7321,.7321,.7325,.7330,.7330,.7330,
     !          .7350,.7370,.7390,.7390,.7393,.7398,.7399,.7399,
     !          .7411,.7432,.7453,.7454,.7457,.7463,.7463,.7463/
      DATA G7/.8069,.8103,.8138,.8139,.8145,.8152,.8153,.8153,
     !          .8454,.8496,.8538,.8540,.8547,.8556,.8557,.8557,
```

```
     !        .8727,.8774,.8822,.8825,.8832,.8842,.8843,.8843,
     !        .8938,.8990,.9042,.9046,.9054,.9064,.9065,.9065,
     !        .9111,.9166,.9222,.9226,.9235,.9246,.9247,.9247,
     !        .9258,.9316,.9374,.9379,.9387,.9399,.9400,.9400,
     !        .9384,.9445,.9506,.9511,.9520,.9532,.9533,.9533,
     !        .9496,.9559,.9622,.9627,.9637,.9649,.9650,.9650,
     !        .9596,.9661,.9727,.9731,.9741,.9754,.9755,.9755,
     !        .9686,.9753,.9821,.9826,.9836,.9849,.9850,.9850,
     !        .9769,.9837,.9906,.9912,.9922,.9936,.9937,.9937,
     !        .9845,.9915,.9986,.9991,.9999,.9999,.9999,.9999,
     !        .9915,.9987,.9999,.9999,.9999,.9999,.9999,.9999,
     !        .9981,.9999,.9999,.9999,.9999,.9999,.9999,.9999/
C-----------------------------------------------------------------
C     REAL AIR EOS, TABLE LOOKUP ON GILMORE DATA.  (NO TEMP. MODEL)
C     TO AVOID COSTLY LOGARITHMIC FUNCTIONS THE TABLE "G" IS STORED IN A
C     FORM SO THAT THE HEXADECIMAL WORD STRUCTURE OF A 32 BIT MACHINE
C     MAY BE EXPLOITED.
C     THIS LOGIC MAY BE TRANSFERED TO OTHER MACHINES BY RECALCULATING
C     THE TABLE "G" APPROPRIATE TO THE WORD ARCITECTURE OF THAT MACHINE.
C     MACHINE DEPENDENT FUNCTIONS AND KEY NUMBERS MUST ALSO BE CHANGED.
C-----------------------------------------------------------------
      RL16E = 1./XL16E
      IST = 0
      NR  = N
C
   10 CONTINUE
      NST = MIN0(NR,M)
C
      DO 20 IRE=1,NST
      RHO(IRE) = .774413*RRR(IST+IRE)
      E(IRE) = AMAX1(3.e8,10000.*EEE(IST+IRE)/RRR(IST+IRE))
C
C     CALCULATE MASS DENSITY VARIATION INDEX "I".
C
      TEM = ALOG(RHO(IRE))*RL16E + 500.0
      I(IRE)   = AINT(TEM)
      OMP(IRE) = TEM - FLOAT(I(IRE))
      I(IRE)   = 502 - I(IRE)
      I(IRE)   = MAX0(I(IRE),1)
C
C     CALCULATE INTERNAL ENERGY VARIATION INDEX "J".
C
      TEM = ALOG(E(IRE))*RL16E
      JCY = AINT(TEM)
      TEM = TEM - FLOAT(JCY)
      TEM = EXP(XL16E*TEM)
      JCY = JCY - 7
      JS  = AINT(TEM)
      Q(IRE)   = TEM - FLOAT(JS)
      J(IRE)   = JS + 15*JCY
      J(IRE)   = MIN0(J(IRE),104)
      J(IRE)   = I(IRE) + 8*J(IRE)
      I(IRE)   = J(IRE) - 8
   20 CONTINUE
C
      DO 30 IRE=1,NST
      T11(IRE) = GF(I(IRE))
      T21(IRE) = GF(I(IRE)+1)
      T12(IRE) = GF(J(IRE))
      T22(IRE) = GF(J(IRE)+1)
   30 CONTINUE
C
C     CALCULATE GAMMA BY LINEAR INTERPOLATION.
C
      DO 40 IRE=1,NST
      T12(IRE) = T12(IRE) - T11(IRE)
      T22(IRE) = T22(IRE) - T21(IRE)
      GAMMA(IST+IRE) =       OMP(IRE) *(T11(IRE) + Q(IRE)*T12(IRE))
     !                 + (1. - OMP(IRE))*(T21(IRE) + Q(IRE)*T22(IRE))
     !                 +  1.
   40 CONTINUE
C
      NR  = NR - NST
      IST = IST + NST
```

```
      IF(NR.GT.0) GO TO 10
C
      RETURN
      END
C
```

Thu Jul 1 14:17:00 1993 threed.f SUBROUTINE MATRLA

```
      SUBROUTINE MATRLA
C
C     READS MATERIAL PROPERTIES
C     EVALUATES MATERIAL RELATED CONTANTS
C
      include      'dmtrl0.h'
      CHARACTER*8   PRODCT(15), PHASE(15), GG, SS, VV, NASAP
      REAL          X(15), KVOL(15), M(15), RHOS(15), CVS(15)
      REAL          CF(7,2), CV(0:5,2)
C
      DATA          KVOL/15*0./
      DATA          GG/'G'/, SS/'S'/, VV/'VVVV'/
      DATA          CV/12*0./
      DATA PRODCT/'O2','N2',13*'VVVV    '/
      DATA PHASE/2*'G',13*'        '/
C
C      DATA PRODCT/'H2O','H2','CO2','CO','NH3','CH4','N2','Cs'
C     1      ,7*'VVVV    '/
C      DATA PHASE/7*'G','S',7*'        '/
C
      ALFAA=.5
      BETAA=.09585
      THETAA=400.
      CAPPAA=12.685
      NNASA=100
      X(1)=21.
      X(2)=79.
      KVOL(1)=350.
      KVOL(2)=380.
      M(1)=32.
      M(2)=28.016
      CVS(1)=0.
      CVS(2)=0.
      RHOS(1)=0.
      RHOS(2)=0.
      NS = 0
      NG = 0
C
      TMS = 0.
      COVA = 0.
      GML = 0.
      SML = 0.
      SV = 0.
      SCVA = 0.
C
       REWIND 4
      DO 110 I = 1, 15
      IF ( PRODCT(I) .EQ. VV ) GO TO 10
      NS = I
C
      IF ( PHASE(I) .EQ. GG ) THEN
      NG = NG + 1
      GML = GML + X(I)
      TMS = TMS + X(I)*M(I)
      COVA = COVA + X(I)*KVOL(I)
C
      ELSE IF ( PHASE(I) .EQ. SS ) THEN
      PHASE(I) = VV
      SML = SML + X(I)
      TMS = TMS + X(I)*M(I)
      SCVA = SCVA + X(I)*CVS(I)
      SV = SV + X(I)*M(I)/RHOS(I)
C
      ELSE
      STOP ' PRODUCTS EITHER SOLID, S, OR GAS, G'
C
```

```
      END IF
110   CONTINUE
C
10    IF ( NS .LT. 1 ) STOP ' NO PRODUCTS ?'
C
      COVA = COVA * CAPPAA / GML
      FSA = TMS/AMAX1(SV,1.E-15)
      TML = GML + SML
      XGA = GML/TML
      SCVA = SCVA/TML
      WMA = TMS/TML
C
      DO 130 INASA = 1, NNASA
      IF ( NG .EQ. 0 ) GO TO 20
1     READ (4,1001) NASAP, ID
1001  FORMAT(A8,71X,I1)
      IF ( ID .NE. 1 ) GO TO 1
C
      DO 120 I = 1, NS
        IF ( NASAP .EQ. PRODCT(I) .AND. PHASE(I) .EQ. GG ) THEN
        PHASE(I) = VV
        NG = NG - 1
        READ (4,1002) ((CF(K,KK),K=1,7),KK=1,2)
1002    FORMAT(5E15.8)
C
          CF(1,1) = CF(1,1) - 1.
          CF(1,2) = CF(1,2) - 1.
          DO 115 K = 0, 5
          CV(K,1) = CV(K,1) + (X(I)/GML)*CF(K+1,1)
115       CV(K,2) = CV(K,2) + (X(I)/GML)*CF(K+1,2)
C
        END IF
120   CONTINUE
C
130   CONTINUE
C
20    DO 140 I = 1, NS
      IF ( PHASE(I) .NE. VV ) STOP ' SPECIES NOT FOUND IN NASA'
140   CONTINUE
C
      DO 150 I = 3, 50
150   TA(I) = FLOAT(100*I)
C
      CALL PSM ( CV(0,2),4, TA(3),8,  CVMA(3)  )
      CALL PSM ( CV(0,1),4, TA(11),40, CVMA(11) )
C
      DO 155 K = 1, 4
      CV(K,1) = CV(K,1)/FLOAT(K+1)
155   CV(K,2) = CV(K,2)/FLOAT(K+1)
C
      CALL PSM ( CV(0,2),4, TA(3),8,  EMEOA(3)  )
      CALL PSM ( CV(0,1),4, TA(11),40, EMEOA(11) )
C
      DO 160 I = 3, 10
160   EMEOA(I) = TA(I)*EMEOA(I)
      DO 161 I = 11, 50
161   EMEOA(I) = TA(I)*EMEOA(I)
C
      DO 180 I = 3, 50
180   EMEOA(I) = EMEOA(I)*XGA + TA(I)*SCVA
C
      CALL BILD (EMEOA,48,RANGEA,DYA)
C
      RETURN
      END
```

```
      SUBROUTINE PSM (A,NPOL, T,N, SMM)
C
      REAL     A(0:NPOL), T(N), SMM(N)
C
      DO 10 J = 1, N
   10 SMM(J) = A(NPOL)
C
      DO 20 K = NPOL-1, 0, -1
C
        DO 15 J = 1, N
   15   SMM(J) = SMM(J) * T(J) + A(K)
C
   20 CONTINUE
C
      RETURN
      END
C
```

Thu Jul 1 14:17:00 1993 threed.f SUBROUTINE BILD

```
      SUBROUTINE BILD(Y,N,RANGE,DY)
C
      REAL Y(N),RANGE,DY(200)
      IF( N .GT. 201 ) STOP ' ONLY 201 POINTS ALLOWED '
C
      RANGE = (Y(N+2) - Y(3)) / (N-1)
      DO 10 I = 1 , N-1
      DY(I+2) = Y(I+3) - Y(I+2)
   10 CONTINUE
C
      RETURN
      END
```

Thu Jul 1 14:17:00 1993 threed.f SUBROUTINE MATRLX

```
      SUBROUTINE MATRLX
C
C     READS MATERIAL PROPERTIES
C     EVALUATES MATERIAL RELATED CONTANTS
C
      include      'dmtrl0.h'
      CHARACTER*8  PRODCT(15), PHASE(15), GG, SS, VV, NASAP
      REAL         X(15), KVOL(15), M(15), RHOS(15), CVS(15)
      REAL         CF(7,2), CV(0:5,2)
C
      DATA         KVOL/15*0./
      DATA         GG/'G'/, SS/'S'/, VV/'VVVV'/
      DATA         CV/12*0./
C
      DATA PRODCT/'H2O','CO2','CO','N2','Cs',10*'VVVV    '/
      DATA PHASE/4*'G','S',10*'        '/
C
      ALFAX=.5
      BETAX=.09585
      THETAX=400.
      CAPPAX=12.685
      NNASA=100
      X(1)=2.5
      X(2)=1.66
      X(3)=.188
      X(4)=1.5
      X(5)=5.15
      KVOL(1)=250.
      KVOL(2)=600.
      KVOL(3)=390.
      KVOL(4)=380.
      KVOL(5)=0.
      M(1)=18.
      M(2)=44.
      M(3)=28.
      M(4)=28.
      M(5)=12.
      CVS(5)=1.1
      RHOS(5)=2.6
```

```
      NS = 0
      NG = 0
C
      TMS = 0.
      COVX = 0.
      GML = 0.
      SML = 0.
      SV = 0.
      SCVX = 0.
C
      REWIND 4
      DO 110 I = 1, 15
      IF ( PRODCT(I) .EQ. VV ) GO TO 10
      NS = I
C
      IF ( PHASE(I) .EQ. GG ) THEN
      NG = NG + 1
      GML = GML + X(I)
      TMS = TMS + X(I)*M(I)
      COVX = COVX + X(I)*KVOL(I)
C
      ELSE IF ( PHASE(I) .EQ. SS ) THEN
      PHASE(I) = VV
      SML = SML + X(I)
      TMS = TMS + X(I)*M(I)
      SCVX = SCVX + X(I)*CVS(I)
      SV = SV + X(I)*M(I)/RHOS(I)
C
      ELSE
      STOP ' PRODUCTS EITHER SOLID, S, OR GAS, G'
C
      END IF
110   CONTINUE
C
10    IF ( NS .LT. 1 ) STOP ' NO PRODUCTS ?'
C
      COVX = COVX * CAPPAX / GML
      FSX = TMS/AMAX1(SV,1.E-15)
      TML = GML + SML
      XGX = GML/TML
      SCVX = SCVX / TML
      WMX = TMS/TML
C
      DO 130 INASA = 1, NNASA
      IF ( NG .EQ. 0 ) GO TO 20
1     READ (4,1001) NASAP, ID
1001  FORMAT(A8,71X,I1)
      IF ( ID .NE. 1 ) GO TO 1
C
      DO 120 I = 1, NS
        IF ( NASAP .EQ. PRODCT(I) .AND. PHASE(I) .EQ. GG ) THEN
        PHASE(I) = VV
        NG = NG - 1
        READ (4,1002) ((CF(K,KK),K=1,7),KK=1,2)
1002    FORMAT(5E15.8)
C
          CF(1,1) = CF(1,1) - 1.
          CF(1,2) = CF(1,2) - 1.
          DO 115 K = 0, 5
          CV(K,1) = CV(K,1) + (X(I)/GML)*CF(K+1,1)
115       CV(K,2) = CV(K,2) + (X(I)/GML)*CF(K+1,2)
C
        END IF
120   CONTINUE
C
130   CONTINUE
C
20    DO 140 I = 1, NS
      IF ( PHASE(I) .NE. VV ) STOP ' SPECIES NOT FOUND IN NASA'
140   CONTINUE
C
      DO 150 I = 3, 50
150   TX(I) = FLOAT(100*I)
C
```

```
      CALL PSM ( CV(0,2),4, TX(3),8,  CVMX(3)  )
      CALL PSM ( CV(0,1),4, TX(11),40, CVMX(11) )
C
      DO 155 K = 1, 4
      CV(K,1) = CV(K,1)/FLOAT(K+1)
  155 CV(K,2) = CV(K,2)/FLOAT(K+1)
C
      CALL PSM ( CV(0,2),4, TX(3),8,  EMEOX(3)  )
      CALL PSM ( CV(0,1),4, TX(11),40, EMEOX(11) )
C
      DO 160 I = 3, 10
  160 EMEOX(I) = TX(I)*EMEOX(I)
      DO 161 I = 11, 50
  161 EMEOX(I) = TX(I)*EMEOX(I)
C
      DO 180 I = 3, 50
  180 EMEOX(I) = EMEOX(I)*XGX + TX(I)*SCVX
C
      CALL BILD (EMEOX,48,RANGEX,DYX)
C
      RETURN
      END
```

Thu Jul 1 14:17:00 1993 threed.f SUBROUTINE VOLMTETC

```
      SUBROUTINE VOLMTETC ( I1, I2, I3, X, Y, Z , VOLUMT )
C
C-----------------------------------------------------------------I
C                                                                 I
C       VOLMTETC FINDS THE VOLUME OF THE TETRAHEDRON DEFINED BY THE   I
C       GRID VERTICES I1, I2, I3, AND THE POINT (X, Y, Z).            I
C       THE CODE ASSUMES THAT THE AREAL VECTOR OF THE BASE TRIANGLE   I
C       FORMED BY I1, I2 AND I3 POINTS IN THE DIRECTION OF (X, Y, Z): I
C       BY THE RIGHT HAND RULE, IF I1, I2 AND I3 ARE ARRANGED         I
C       COUNTER-CLOCKWISE AS VIEWED FROM ABOVE THE PLANE OF THE       I
C       TRIANGLE, (X, Y, Z) ALSO LIES ABOVE THE PLANE).  BUT NOTE --  I
C                                                                 I
C                THE VOLUME RETURNED IS A SIGNED QUANTITY - IE.       I
C                IF THE VERTICES ARE NOT ORDERED BY THE RIGHT         I
C                HAND RULE THE VOLUME WILL BE NEGATIVE.               I
C                                                                 I
C                                                                 I
C       DECEMBER, 1991: M. FRITTS, FRITTS%MCL.SAINET@CCC.NERSC.GOV,   I
C                                  (301) 266-0992                     I
C                                                                 I
C-----------------------------------------------------------------I
C
C=================================================================
C
       DOUBLE PRECISION R21X,R21Y,R21Z,R31X,R31Y,R31Z,R41X,R41Y,R41Z
       DOUBLE PRECISION VOLUMT,X,Y,Z
C
C
      include    'dmsh00.h'
C
C=================================================================
C
C --- FIND THE VOLUME OF THE TETRAHEDRON.------------------------------
C
        R21X = XV(1,I2) - XV(1,I1)
        R21Y = XV(2,I2) - XV(2,I1)
        R21Z = XV(3,I2) - XV(3,I1)
        R31X = XV(1,I3) - XV(1,I1)
        R31Y = XV(2,I3) - XV(2,I1)
        R31Z = XV(3,I3) - XV(3,I1)
        R41X = X         - XV(1,I1)
        R41Y = Y         - XV(2,I1)
        R41Z = Z         - XV(3,I1)
C
        VOLUMT  = ( R41X*( R21Y*R31Z - R21Z*R31Y ) -
     1              R41Y*( R21X*R31Z - R21Z*R31X ) +
     2              R41Z*( R21X*R31Y - R21Y*R31X )   )/6.D0
C
C
```

```
C        ------
         RETURN
C        ------
C
C     ---
      END
C
```

Thu Jul 1 14:15:40 1993 mainhd.f Module List - alphabetical order

```
      PROGRAM AUGUST
C=================================================================
C
C      The AUGUST Code
C
C                     -  Adaptive
C                     -  Unstructured
C                     -  Godunov
C                     -  Upwind
C                     -  Second order
C                     -  Triangular
C
C                     The geometry structure comes from BERMUDA
C                     The solver is based on FUGGS
C
C
C      Version:   2.00      june 17, 1991
C
C      Authors:   Itzhak Lottati    (703)749-8648
C                 Shmuel Eidelman   (703)448-6491
C                 Adam Drobot       (703)734-5840
C
C                 Science Applications International Corporation
C                 Applied Physics Operation
C                 1710 Goodridge Drive
C                 McLean, Virginia 22102
C
C=================================================================
C
C
C-------------------------------------------------------------I
C                                                             I
C     BERMUDA IS A MULTIDIMENSIONAL CODE WHICH IS BASED ON THE I
C             USE OF TRIANGULAR GRIDS AS THE FUNDAMENTAL MESH I
C             FOR FIELD LIKE QUANTITIES.  THE CODE REQUIRES   I
C             THAT ALL QUANTITIES ARE BASED AT THE BARICENTER I
C             OF SIDES/TRIANGLES.                             I
C                                                             I
C             THE QUIP IS THAT THOSE WHO WORK ON BERMUDA      I
C             TRIANGLES ARE NEVER HEARD FROM AGAIN.           I
C                                                             I
C             THE BASIC MODULES IN BERMUDA INCLUDE:           I
C                                                             I
C                    A HYDRODYNAMICS CODE                     I
C                          .BASED ON A FIRST ORDER GODUNOV    I
C                           METHOD OR A SECOND ORDER GODUNOV  I
C                           WITH MESH ADAPTATION.             I
C                                                             I
C-------------------------------------------------------------I
C
C     GRID SETUP TABLES AND THEIR MEANING:
C
C     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C     +                                                        +
C     +   LIST OF VERTICES                                     +
C     +                                                        +
C     +        IV          - VERTEX INDEX                      +
C     +        JV(1,IV)    - S STATUS OF THE POINT             +
C     +                      S=0 FREE POINT WHICH MAY BE       +
C     +                          DELETED/MOVED                 +
C     +                      S=1 POINT RESTRICTED TO A SURFACE +
C     +                      S=2 POINT RESTRICTED TO A LINE    +
C     +                      S=3 POINT WHICH IS FIXED AND MAY  +
C     +                          NOT BE REMOVED                +
C     +        JV(2,IV)    - INDEX OF A LINE WHICH INCLUDE THE +
C     +                      POINT                             +
C     +                      NEGATIVE MEANS THE POINT IS ON A  +
C     +                      BOUNDARY LINE                     +
C     +        XV(1,IV)    - X POSITION OF VERTEX              +
C     +        XV(2,IV)    - Y POSITION OF VERTEX              +
C     +                                                        +
C     ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
```

```
C       ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C       +                                                          +
C       +   LIST OF EDGES                                          +
C       +                                                          +
C       +       IE          - EDGE INDEX                           +
C       +       JE(1,IE)    - INDEX OF LOWER EDGE VERTEX           +
C       +       JE(2,IE)    - INDEX OF UPPER EDGE VERTEX           +
C       +                                                          +
C       +   FOR TWO DIMENSIONAL PROBLEMS                           +
C       +                                                          +
C       +       JE(3,IE)    - INDEX OF LEFT  SIDE                  +
C       +       JE(4,IE)    - INDEX OF RIGHT SIDE                  +
C       +                                                          +
C       +       IF JE(3-4,IE) IS NEGATIVE THIS INDICATES THAT THE +
C       +                     EDGE LIES ALONG A BOUNDARY.          +
C       +                                                          +
C       +       JE(5,IE)    - INDEX DEFINING BOUNDARY CONDITION    +
C       +                     0 - ORDINARY EDGE INTERIOR           +
C       +                     6 - WALL V-PERPENDICULAR=0           +
C       +                     7 - SUPERSONIC OUTFLOW               +
C       +                     8 - INFLOW BOUNDARY                  +
C       +                                                          +
C       +       XE(1,IE)    - LENGTH OF EDGE                       +
C       +       XE(2,IE)    - DISTANCE BETWEEN ADJOINING SIDE      +
C       +                     POINTS.                              +
C       +                                                          +
C       ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C       ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C       +                                                          +
C       +   LIST OF SIDES                                          +
C       +                                                          +
C       +       IS          - SIDE INDEX                           +
C       +       JS(1,IS)    - INDEX OF FIRST  VERTEX               +
C       +       JS(2,IS)    - INDEX OF SECOND VERTEX               +
C       +       JS(3,IS)    - INDEX OF THIRD  VERTEX               +
C       +                                                          +
C       +       THE VERTICES RUN AROUND THE SIDE IN ORDER          +
C       +       COUNTER-CLOCKWISE FASHION                          +
C       +                                                          +
C       +       JS(4,IS)    - INDEX OF THE FIRST  EDGE             +
C       +       JS(5,IS)    - INDEX OF THE SECOND EDGE             +
C       +       JS(6,IS)    - INDEX OF THE THIRD  EDGE             +
C       +                                                          +
C       +       THE EDGES ARE ARRANGED IN COUNTER-iCLOCKWISE       +
C       +       FASHION. EDGE ONE RUNS FROM VERTEX-ONE TO          +
C       +       VERTEX-TWO ETC.. THE SIGN OF JS(4-6,IS) INDICATES +
C       +       IF EDGE DATA IS STORED THE SAME WAY. IF IT IS      +
C       +       JS>0 AND IT IS REVERSED JS<0                       +
C       +                                                          +
C       +       XS(1,IS)     - X POSITION OF SIDE POINT            +
C       +       XS(2,IS)     - Y POSITION OF SIDE POINT            +
C       +       XS(3,IS)     - AREA OF SIDE                        +
C       +                                                          +
C       +                                                          +
C       ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C======================================================================
C
C --- DEFINITION FOR ALL HYDRODYNAMIC QUANTITIES ---------------------
C
C-------------------------------------------------------I
C                                                       I
C     USE OF PARAMETERS:                                I
C                                                       I
C         MHQ     - MAXIMUM NUMBER OF HYDRO QUANTITIES. I
C                                                       I
C                                                       I
C-------------------------------------------------------I
C
C======================================================================
C
      include     'cmsh00.h'
      include     'chyd00.h'
```

```
      include    'cint00.h'
      include    'cphs10.h'
      include    'cphs20.h'
C
C=====================================================================
C
      NAMELIST /DATA/ ICOND,ICONP,ITRIGR,IOPTN,
     .                XMCHIN,RIN,PIN,ALFA,HRGG,IHRN,NTIME,MDUMP,NDUMP,
     .                KDUMP,IOSPCL,IOPLFT,IOPRCN,IOPORD,IOPBYN,IAXSYM,
     .                IOPEOS,MPRTCL,IOPINT,IOPADD,IOPDEL,AREADD,AREDEL,
     .                IWINDW,ISTATC
C
C--------------------------------------------------------------------
C
C
C     --- MEANING OF NAMELIST VARIABLES:                                 I
C                                                                         I
C     ICOND = 0 READ INPUT GRID FOR A NEW RUN                            I
C           = 1 READ THE GRID FROM PREVIOUS RUN                          I
C     ICONP = 0 PRIMITIVE VARIABLES SET TO ZERO                          I
C           = 1 VARIABLES READ FROM PREVIOUS RUN                         I
C     ITRIGR = 0 USING THE INPUT GRID AS THE INITIAL GRID                I
C            = 1 THE INPUT GRID TRIPLED BY ADDING AN EXTRA VERTEX IN     I
C                EACH TRIANGLE                                           I
C     IOPTN = 1 SOLUTION FOR STEADY STATE,                               I
C           = 2 SOLUTION FOR TRANSIENT PHENOMENA                         I
C                                                                         I
C     XMCHIN = FOR TRANSIENT SHOCK CALCULATIONS(IOPTN=2)THIS VARIABLE    I
C              IS USED TO SPECIFY THE UPSTREAM MACH NUMBER               I
C                                                                         I
C              RIN = THE AMBIENT DENSITY IN THE CHAMBER                  I
C                                                                         I
C              PIN = THE AMBIENT PRESSURE IN THE CHAMBER                 I
C                                                                         I
C              APPLYING NORMAL SHOCK WAVES RELATIONS FOR AN ADIABATIC    I
C              FLOW RELATION STATIC-PRESSURE RATIO ACROSS THE SHOCK      I
C              AS WELL AS THE DENSITY RATIO AND MACH NUMBER RATIO        I
C              ARE COMPUTED TO SET CORRECTLY THE CONDITION AT THE        I
C              INLET EDGES( EDGE BOUNDARY 8 ) OF THE COMPUTATIONAL       I
C              DOMAIN                                                    I
C                                                                         I
C              FOR STEADY STATE SHOCK CALCULATIONS(IOPTN=1)THIS IS THE   I
C              INFLOW MACH NUMBER. ALL DOMAIN VELOCITIES ARE THEN        I
C              INITIALIZED WITH THIS VALUE.                              I
C                                                                         I
C              RIN = THE AMBIENT DENSITY AT INFINITY                     I
C                                                                         I
C              PIN = THE AMBIENT PRESSURE AT INFINITY                    I
C                                                                         I
C              ALL COMPUTATIONAL DOMAIN ARE THEN INITIALIZED WITH        I
C              THOSE VALUES.                                             I
C                                                                         I
C     ALFA   = THE DIRECTION OF INFLOW IN DEGREES RELATIVE TO A RIGHT    I
C              HAND COORDINATE SYSTEM. ALFA=0 MEANS FLOW FROM LEFT TO    I
C              RIGHT. ALFA=90 MEANS FROM BOTTOM TO TOP. ALFA=-90 OR 270 I
C              MEANS FLOW FROM TOP TO BOTTOM ETC.                        I
C     HRGG   = INITIAL GAMMA IN THE EQUATION OF STATE                    I
C              THE CODE RUNS USING THE AIR EQUATION AS A BASELINE AND    I
C              SHOULD BE MODIFIED IF SOMETHING ELSE IS DESIRED.          I
C     IHRN   = NUMBER OF ITERATIONS IN THE RIEMANN SOLVER TO FIND THE    I
C              DIAPHRAGM SOLUTION.(3 to 4 SHOULD BE USED AND INCREASED   I
C              ONLY FOR HIGH MACH NUMBER CASES).                         I
C                                                                         I
C     NTIME = NUMBER OF REPEATS FOR THE INTEGRATION/REFINEMENT/          I
C              COARSENING SEQUENCE                                       I
C              AN OUTPUT DUMP IS DONE EVERY SEQUENCE REPEAT.             I
C     MDUMP = NUMBER OF OUTERMOST LOOP ITERATIONS IN THE CALCULATION     I
C              WHERE COARSENING OF THE GRID IS PERFORMED EVERY SEQUENCE I
C              REPEAT.                                                   I
C     NDUMP = NUMBER OF OUTER LOOP ITERATIONS IN THE CALCULATION WHERE I
C              REFINING IS DONE EVERY SEQUENCE REPEAT WITHOUT COARSENINGI
C     KDUMP = NUMBER OF ITERATIONS PERFORMED WITH NO REFINEMENT OR      I
C              COARSENING. IT IS THE INNER LOOP OF THE CALCULATION.      I
C              IF KDUMP=0 IS READ IN, KDUMP WILL BE SET BY THE           I
```

```
C             CODE AUTOMATICALLY ACCORDING TO THE VARIABLE AREADD.      I
C                                                                       I
C       +___________________ o NTIME - DUMPING DATA                     I
C       I                                                               I
C       I    +______________ o MDUMP - COARSENING                       I
C       I    I                                                          I
C       I    I    +_________ o NDUMP - REFINENEMENT                     I
C       I    I    I                                                     I
C       I    I    I    +____ o KDUMP - INTEGRATION                      I
C       I    I    I    I                                                I
C       I    I    I    I                                                I
C       I    I    I    +____ o ..... INNER LOOP                         I
C       I    I    I                                                     I
C       I    I    +_________ o ..... OUTER LOOP                         I
C       I    I                                                          I
C       I    +______________ o ..... OUTERMOST LOOP                     I
C       I                                                               I
C       +___________________ o ..... DUMPING LOOP                       I
C                                                                       I
C      IOSPCL = 0 NOT USING REDEFNITION OF POINTS ON THE BOUNDARY       I
C             = 1 USING REDEFENITION OF POINTS ON THE BOUNDARY          I
C      IOPLFT = 0 THE COMPUTATION OF LIFT DRAG AND MOMENT TURNED OFF    I
C             = 1 THE COMPUTATION OF LIFT DRAG AND MOMENT TURNED ON     I
C      IOPRCN = 0 A GLOBAL SWAPING ( RECONNECTION ) PROCEDURE IS OFF    I
C             = 1 A GLOBAL SWAPING ( RECONNECTION ) PROCEDURE IS ON     I
C      IOPORD = 1 THE CODE WILL RUN FIRST ORDER GODUNOV METHOD          I
C             = 2 THE CODE WILL RUN SECOND ORDER GODUNOV METHOD         I
C      IOPBYN = 0 NO BUOYANCY EFFECT ARE COMPUTED                       I
C             = 1 BUOYANCY EFFECT IN THE Y DIRECTION ARE COMPUTED       I
C      IAXSYM = 0 THE CODE WILL RUN IN A PURE TWO DIMENSIONAL MODE      I
C             = 1 THE CODE WILL RUN IN AN AXI SYMMETRICAL MODE (X AXIS) I
C             = 2 THE CODE WILL RUN IN AN AXI SYMMETRICAL MODE (Y AXIS) I
C      IOPEOS = 0 THE CODE WILL RUN WITH CONSTANT GAMA                  I
C             = 1 THE CODE WILL RUN WITH VARIABLE GAMA USING EQUATION   I
C                 OF STATE FOR AIR                                      I
C                                                                       I
C      MPRTCL = 0 NO PARTICLE TRACING                                   I
C             = 1 THE CODE WILL TRACE PARTICLES                         I
C      IOPINT = 0 NOT REFINING INITIALY THE EDGE BOUNDARY NO 8          I
C             = 1 REFINING INITIALY THE EDGE BOUNDARY NO 8              I
C      IOPADD = 0 THE REFINEMENT PROCEDURE IS TURNED OFF                I
C             = 1 THE REFINEMENT PROCEDURE IS TURNED ON                 I
C      IOPDEL = 0 THE COARSENING PROCEDURE IS TURNED OFF                I
C             = 1 THE COARSENING PROCEDURE IS TURNED ON                 I
C      AREADD = SPECIFY THE MINIMUM VALUE THAT A TRIANGLE SHOULD HAVE   I
C               AFTER REFINEMENT AS A FRACTION OF AVERAGE TRIANGLE AREA I
C      AREDEL = SPECIFY THE MAXIMUM VALUE THAT A TRIANGLE SHOULD HAVE   I
C               AFTER COARSENING AS A FRACTION OF AVERAGE TRIANGLE AREA I
C                                                                       I
C      IWINDW = 0 NO RESTRICTION ON THE REGION FOR REFINING THE GRID    I
C             = 1 SETTING A WINDOW FOR REFINING THE GRID                I
C      ISTATC = 0 THE ADAPTATION WILL BE DONE ON A MOVING WAVE          I
C             = 1 THE ADAPTATION WILL BE DONE ON A STEADY STATE         I
C                 CONDITION                                             I
C                                                                       I
C-----------------------------------------------------------------------I
C
       CHARACTER*15 ZHEADER,MNAME,MVNAME
       CHARACTER*1  FILLCH
       INTEGER NUMQUADS
C
C --- OPEN ALL FILES FOR THIS RUN ----------------------------------------
C
       OPEN( 4,FILE='naca4'  ,FORM='UNFORMATTED')
       OPEN(88,FILE='naca82' ,FORM='UNFORMATTED')
       OPEN( 8,FILE='naca2'  ,FORM='UNFORMATTED')
       OPEN( 9,FILE='naca3'  ,FORM='UNFORMATTED')
       OPEN( 2,FILE='data.d',FORM='FORMATTED')
       OPEN(16,FILE='wedge45.zon',STATUS='OLD')
       OPEN(18,FILE='nacaa'  ,FORM='UNFORMATTED')
C
C========================================================================
C
C --- DEFAULT VALUES FOR INPUT DATA   ------------------------------------
```

```
C
      THIRD = 1. / 3.
C
      ICOND = 0
      ICONP = 0
      ITRIGR = 0
      IOPTN = 1
C
      XMCHIN = 25.
      RIN = 1.
      PIN = 1.
C
      ALFA = 0.
      HRGG = 1.4
      IHRN = 4
      NTIME = 1
      MDUMP = 80
      NDUMP = 1
      KDUMP = 0
      IOSPCL = 0
      IOPLFT = 0
      IOPRCN = 0
      IOPORD = 2
      IOPBYN = 0
      IAXSYM = 0
      IOPEOS = 0
C
      MPRTCL = 0
      IOPINT = 0
      IOPADD = 0
      IOPDEL = 0
      AREADD = 0.005
      AREDEL = 1.
      IWINDW = 0
      ISTATC = 0
C
C --- READ THE INPUT DATA ---------------------------------------------
C
      READ (2,DATA)
C
C --- PRINTOUT THE RUN PARAMETERS -------------------------------------
C
        PRINT 101,     ICOND,ICONP,ITRIGR,IOPTN,
     .                 XMCHIN,RIN,PIN,ALFA,HRGG,IHRN,NTIME,MDUMP,NDUMP,
     .                 KDUMP,IOSPCL,IOPLFT,IOPRCN,IOPORD,IOPBYN,IAXSYM,
     .                 IOPEOS,MPRTCL,IOPINT,IOPADD,IOPDEL,AREADD,AREDEL,
     .                 IWINDW,ISTATC
C
C --- SET RUN CONDITIONS AND PRINTOUT TO CONSOLE ----------------------
C
      XREADD = 1. / AREADD
      NAREAD = ALOG( XREADD ) / ALOG( 3. ) + 1
      IF( NAREAD . LT . 3 ) NAREAD = 3
      IF( NAREAD . GT . 5 ) NAREAD = 5
      IF( ISTATC . EQ . 1 ) NAREAD = 3
      PRINT*,AREADD,AREDEL,NAREAD
      PRINT * ,ICOND,ICONP
C
      NPT = 0
      IJKINT = 3
      IF( ICOND . EQ . 0 ) THEN
      DO 122 IS = 1 , MSM
      KSDELT( IS ) = 0
  122 CONTINUE
      END IF
      HYDMOM( 1 ) = 0.
      HYDMOM( 2 ) = 0.
      HYDMOM( 4 ) = 0.
C
      DO 124 IK = 1 , MBP
      GAMAG( IK ) = HRGG
  124 CONTINUE
C
C=====================================================================
```

```
C
C --- READ IN THE MESH DATA -------------------------------------------
C
C(1)>>>>>
         IF( ICOND . EQ . 0 ) THEN
         IF( ICONP . EQ . 1 ) CALL UPGRAD
C
C----------------------------------------------------------------
C A "SMART" FORMAT MESH FILE IS READ. THE FILE IS SELECTED BY THE
C NORMAL MACINTOSH FILE DIALOG BECAUSE OF THE '*' IN PLACE OF THE
C FILE NAME. VERTICES OF EACH TRIANGLE ARE FORMED FROM THE INPUT.
C----------------------------------------------------------------
      READ (16,900) ZHEADER
900   FORMAT(A15)
      IF (ZHEADER .NE. 'SMaRT-Z-T-(003)') THEN
C------------------------ THIS ROUTINE CANNOT READ ANY OTHER INPUT
      PRINT *, 'MESH FILE IS NOT THE CORRECT KIND OR VERSION'
      CALL EXIT
      ENDIF
      READ (16,910) FILLCH,NV,NVMK
      PRINT *,NV,NVMK
      READ (16,910) FILLCH,NE,NEMK
      PRINT *,NE,NEMK
      READ (16,910) FILLCH,NS
      PRINT *,NS
      READ (16,910) FILLCH,NUMQUADS
      PRINT *,NUMQUADS
910   FORMAT(A1,2I7)
      READ (16,920) FILLCH,NZMK,NSMK,NNMK
      PRINT *,NZMK,NSMK,NNMK
920   FORMAT(A1,1X,3I3)
      IF (NV .GT. MVM) THEN
C-------------------------- CHECK NODE (I.E., VERTEX) STORAGE SIZE
      PRINT 1020,NV,MVM,NVMK
1010  FORMAT(1X,'TOO MANY NODES. ',I9,', MAX = ',I5)
      CALL EXIT
      ENDIF
      IF (NE .GT. MEM) THEN
C----------------------------- CHECK SIDE (I.E., EDGE) STORAGE SIZE
      PRINT 1020,NE,MEM,NEMK
1020  FORMAT(1X,'TOO MANY SIDES. ',I9,', MAX = ',I5)
      CALL EXIT
      ENDIF
      IF (NS .GT. MSM) THEN
C---------------- CHECK ZONE (I.E., SIDE OR TRIANGLE) STORAGE SIZE
      PRINT 1030,NS,MSM
1030  FORMAT(1X,'TOO MANY ZONES. ',I9,', MAX = ',I5)
      CALL EXIT
      ENDIF
      IF (NUMQUADS .GT. 0) THEN
C---------------------------- CHECK FOR QUADRILATERALS IN THE INPUT
      PRINT 1040
1040  FORMAT(1X,'NO QUADRILATERALS ARE ALLOWED.')
      CALL EXIT
      ENDIF
C------------------ READ MARKER DEFINITIONS
C                   THE FOLLOWING JUST READS THE VARIABLES WITHOUT STORING
C                   THEM INTO PERMANENT ARRAYS, EFFECTIVELY JUST READING
C                   PAST THE MARKER DEFINTION INFORMATION.
      DO 21 NZM = 1,NZMK
      READ (16,1050) NMN,MNAME,NVAL
      DO 20 NZMV = 1,NVAL
      READ (16,1050) NMV,MVNAME
20    CONTINUE
21    CONTINUE
1050  FORMAT(3X,I2,1X,A15,1X,I2)
      DO 31 NZM = 1,NSMK
      READ (16,1050) NMN,MNAME,NVAL
      DO 30 NSMV = 1,NVAL
      READ (16,1050) NMV,MVNAME
30    CONTINUE
31    CONTINUE
      DO 41 NZM = 1,NNMK
      READ (16,1050) NMN,MNAME,NVAL
```

```
      DO 40 NNMV = 1,NVAL
      READ (16,1050) NMV,MVNAME
40    CONTINUE
41    CONTINUE
C------------------ READ IN VERTEX INFORMATION
      DO 51 IV = 1 , NV
      IS = IV
      READ (16,1210) IK,XV(1,IS),XV(2,IS)
      JV(1,IV) = 0
C     INITIALIZE ANY VERTEX MARKER STORAGE, I.E. JV(*,IV)
51    CONTINUE
      PRINT 1060,NV
1060  FORMAT(I5,' NODES (VERTICES) READ IN.')
1210  FORMAT(I7,E15.9,1X,E15.9)
      IF (NVMK .GT. 0) THEN
C------------------ READ IN VERTEX MARKER INFORMATION
      DO 55 IV = 1,NVMK
      READ (16,*) IXV,MV1,MV2,MV3,MV4
      JV(1,IXV) = MV1
C            STORE THESE MARKERS IN JV(*, IXV) AS DESIRED
55    CONTINUE
      PRINT 1070,NVMK
1070  FORMAT(I5,' NODE (VERTEX) MARKERS READ IN.')
      ENDIF
C------------------ READ IN EDGE INFORMATION ( EDGES OF TRIANGLES).
      DO 60 IE = 1 , NE
      IS = IE
      READ (16,*) IJ,JE(1,IS),JE(2,IS),JE(3,IS),JE(4,IS)
C     INITIALIZE ANY MARKER STORAGE.
      JE(5,IE) = 0
60    CONTINUE
      PRINT 1080,NE
1080  FORMAT(I5,' SIDES (EDGES) READ IN.')
      IF (NEMK .GT. 0) THEN
C------------------ READ IN EDGE MARKER INFORMATION
      DO 65 IV = 1,NEMK
      READ (16,*) IXE,MV1,MV2,MV3,MV4
      JE(5,IXE) = MV1
65    CONTINUE
      PRINT 1090,NEMK
1090  FORMAT(I5,' SIDE (EDGE) MARKERS READ IN.')
      ENDIF
C------------------ READ IN SIDE (TRIANGLE) INFORMATION.
      DO 81 IS = 1 , NS
      IE = IS
      READ (16,1100) IJ, MV1,MV2,MV3,MV4,
     .               IV1,ID1,IV2,ID2,IV3,ID3
1100  FORMAT (I7,4I3,3(I7,I2))
      JS(4,IE) = IV1 * ID1
      JS(5,IE) = IV2 * ID2
      JS(6,IE) = IV3 * ID3
C      JS(7,IE) = MV1
C
C     STORE THESE MARKERS IN JS(*, IS) AS DESIRED
C
81    CONTINUE
      PRINT 1110,NS
1110  FORMAT(I5,' ZONES (SIDES) READ IN.')
      CLOSE (16)
C------------------ FORM VERTEX INDICES FOR EACH SIDE (TRIANGLE).
      DO 85 IS = 1 , NS
      DO 85  J = 1 , 3
      IE = JS( J + 3 , IS )
      IEABS = IABS( IE )
      IF ( IE . GT . 0 ) THEN
      JS( J , IS ) = JE( 1 , IEABS )
      ELSE
      JS( J , IS ) = JE( 2 , IEABS )
      END IF
85    CONTINUE
C
C------------------------------------------------------------------
C
      IF(IOSPCL.EQ.1)THEN
```

```
C
C --- SPECIAL CASE FOR HALF CIRCLE  BOUNDARY DATA ---------------------
C
        DO 382 IE = 1 , NE
        IJE5 = JE( 5 , IE )
        IF( IJE5 . EQ . 6 ) THEN
C
        IV1 = JE( 1 , IE )
        IV2 = JE( 2 , IE )
C
        XXS1 = XV( 1 , IV1 )
        YYS1 = XV( 2 , IV1 )
        XXS2 = XV( 1 , IV2 )
        YYS2 = XV( 2 , IV2 )
        DXX = XXS1 - 1.50
        ANGL = 1.570796327
        IF( DXX . NE . 0 ) ANGL = ATAN2( YYS1 , DXX )
        XV( 1 , IV1 ) = COS( ANGL ) + 1.5
        XV( 2 , IV1 ) = SIN( ANGL )
        DXX = XXS2 - 1.50
        ANGL = 1.570796327
        IF( DXX . NE . 0 ) ANGL = ATAN2( YYS2 , DXX )
        XV( 1 , IV2 ) = COS( ANGL ) + 1.5
        XV( 2 , IV2 ) = SIN( ANGL )
C        XXS = XV( 1 , IV1 ) * 1.008930411364
C        YYS = .6 * ( .2969 * SQRT( XXS ) - .126 * XXS -
C     .              .3516 * XXS * XXS + .2843 * XXS * XXS * XXS -
C     .              .1015 * XXS * XXS * XXS * XXS )
C        XV( 2 , IV1 ) = SIGN( 1. , XV( 2 , IV1 ) ) * YYS
C        IF( XXS . GT . .3 . AND . XXS . LT . .7 ) JV( 1 , IV1 ) = 0
C
C        XXS = XV( 1 , IV2 ) * 1.008930411364
C        YYS = .6 * ( .2969 * SQRT( XXS ) - .126 * XXS -
C     .              .3516 * XXS * XXS + .2843 * XXS * XXS * XXS -
C     .              .1015 * XXS * XXS * XXS * XXS )
C        XV( 2 , IV2 ) = SIGN( 1. , XV( 2 , IV2 ) ) * YYS
C        IF( XXS . GT . .3 . AND . XXS . LT . .7 ) JV( 1 , IV2 ) = 0
C        IF( XE( 1 , IE ).GT. .2 ) CALL DISECT ( IE , IDONE , IJKINT )
        END IF
382    CONTINUE
        END IF
C
C====================================================================
C
C
C --- CALCULATE GRID QUANTITIES THROUGH GEOMTR ------------------------
C
        CALL UPDATE
C
C====================================================================
C
C --- REFINE THE INITIAL GRID BY A FACTOR OF THREE IF CALLED FOR -------
C
C>>>>>>
        IF( ITRIGR . EQ . 1 ) THEN
        NSS = NS
        DO 110 IS = 1 , NSS
          CALL VERCEN( IS )
110     CONTINUE
        NEE = NE
        DO 120 IE = 1 , NEE
          IF( JE( 5 , IE ) . NE . 0 ) THEN
             CALL DISECT ( IE , IDONE , IJKINT )
          ENDIF
120     CONTINUE
        DO 130 IK = 1 , 3
        PRINT*,NV,NE,NS,IK
          DO 130 IE = 1 , NE
            CALL RECNC( IE , IDONE , ITL , ITR , JA , JB , JC , JD )
            CALL RECNC( JA , JADONE , ITL , ITR , JAA , JAB , JAC , JAD )
            CALL RECNC( JB , JBDONE , ITL , ITR , JBA , JBB , JBC , JBD )
            CALL RECNC( JC , JCDONE , ITL , ITR , JCA , JCB , JCC , JCD )
            CALL RECNC( JD , JDDONE , ITL , ITR , JDA , JDB , JDC , JDD )
130     CONTINUE
```

```
        END IF
C<<<<<<
C
C=================================================================
C
C --- FIND AVERAGE TRIANGLE AREA -----------------------------------
C
      SAREMN = 1000000.
      SAREMX = 0.
      SAREVG = 0.
      DO 105 IS = 1 , NS
       AREASS = XS( 3 , IS )
       SAREMX = AMAX1( SAREMX , AREASS )
       SAREMN = AMIN1( SAREMN , AREASS )
       SAREVG = SAREVG + AREASS
  105 CONTINUE
      AVGARE = SAREVG
      SAREVG = SAREVG / NS
      FMINVG = SAREVG * AREADD
      SAREMN = SAREMN / SAREVG
      SAREMX = SAREMX / SAREVG
      PRINT*,SAREVG,SAREMX,SAREMN
C
C --- DO INITIAL REFINEMENT FOR ALL INFLOW BOUNDARIES DEFINED ----------
C     BY EDGES THAT CONTAIN BOUNDARY CONDITION 8(INFLOW)
C
      IF(IOPINT.EQ.1)THEN
      NOFDIV = 2
      CALL INTPTN( AREADD , NOFDIV , 1 , LTRIG )
      NOFDIV = 2
      CALL DYYPTN( AREADD , NOFDIV , 1 , LTRIG )
      NOFDIV = 2
      CALL INTPTN( AREADD , NOFDIV , 2 , LTRIG )
      NOFDIV = 2
      CALL DYYPTN( AREADD , NOFDIV , 2 , LTRIG )
      NOFDIV = 2
      CALL INTPTN( AREADD , NOFDIV , 3 , LTRIG )
      NOFDIV = 2
      CALL DYYPTN( AREADD , NOFDIV , 3 , LTRIG )
C
      PRINT*,NV,NE,NS
      ENDIF
C
C=================================================================
C
C --- FOR ICOND>0 READ IN PREVIOUS RUN'S DATA --------------------------
C
C(1)----
      ELSE
        CALL UPGRAD
C         CALL GEOMTR
       IF( ICONP . EQ . 0 ) THEN
         READ (88) RIN,PIN,RINL,PINL,UVIN,UIN,VIN,TT,
     .              HYDMOM(1),HYDMOM(2),HYDMOM(4)
         PRINT *, RIN,PIN,UVIN,UIN,VIN,TT
         READ (88) ((HYDV(IS,IK),IK=1,5),IS=1,NS)
         READ (88) ((HYDVVV(IV,IK),IK=1,5),IV=1,NV)
         READ (88) IJKINT,(KSDELT(IS),IS=1,NS)
          IF( MPRTCL . EQ . 1 )
     .       READ (88) NPT,((XPRTCL(IK,IPT),IK=1,2),IPT=1,NPT),
     .                               (IJKPRT(IPT),IPT=1,NPT)
      ENDIF
      ENDIF
C(1)<<<<
C
C --- INITIALIZATION OF THE PROBLEM ----------------------------------
C
      SARERV = 1. / SAREVG
      SARESQ = SQRT( SAREVG )
      FMINVG = SAREVG * AREADD
      HRSM = 1.E-8
      HRGP = HRGG + 1.
      HRGM = HRGG - 1.
      CF   = HRGP / ( 2. * HRGG )
```

```
C
      JDUMP = 9
      IF(KDUMP.EQ.0)THEN
      KDUMP = JDUMP
      ENDIF
C
      TT = 0.
C
      PIRAD = ATAN( 1. ) / 45.
      ALPHA = ALFA * PIRAD
      PRINT *,ALFA,PIRAD,ALPHA
      PRINT *,XMCHIN,PIN,RIN
C
      COSS = COS( ALPHA )
      SINN = SIN( ALPHA )
      TANN = TAN( ALPHA )
C
C=====================================================================
C
C --- SET THE INITIAL VALUE FOR PRIMITIVE VARIABLES --------------------
C
C(2)>>>>
      IF( IOPTN . EQ . 1 ) THEN
      UVIN = XMCHIN * SQRT( HRGG * PIN / RIN )
      UIN = UVIN * COSS
      VIN = UVIN * SINN
      RIN = 1.
      PIN = 1.
C
      DO 150 IS = 1 , NS
      HYDV( IS , 1 ) = RIN
      HYDV( IS , 2 ) = 0.
      HYDV( IS , 3 ) = 0.
      HYDV( IS , 4 ) = PIN
      HYDV( IS , 5 ) = HRGG
      XSS = XS( 1 , IS )
      IF( XSS . LT . .0 ) THEN
      HYDV( IS , 1 ) = .125 * RIN
      HYDV( IS , 4 ) = .100 * PIN
      END IF
  150 CONTINUE
C
      DO 176 IV = 1 , NV
      HYDVVV( IV , 1 ) = RIN
      HYDVVV( IV , 2 ) = 0.
      HYDVVV( IV , 3 ) = 0.
      HYDVVV( IV , 4 ) = PIN / HRGM
      HYDVVV( IV , 5 ) = HRGG
      XSS = XV( 1 , IV )
      IF( XSS . LT . -.0 ) THEN
      HYDVVV( IV , 1 ) = RIN
      HYDVVV( IV , 4 ) = PIN / HRGM
      END IF
  176 CONTINUE
C(2)----
      ELSE
C
      XMSQR = XMCHIN * XMCHIN
      IF( ICOND . EQ . 1 . AND . ICONP . EQ . 0 ) THEN
      ELSE
      PINL = PIN
      RINL = RIN
      RINRTO = ( HRGG + 1. ) * XMSQR /
     .         ( ( HRGG - 1. ) * XMSQR + 2. )
      PINRTO = ( 2. * HRGG * XMSQR - ( HRGG - 1. ) ) /
     .                                 ( HRGG + 1. )
      PIN = PINRTO * PINL
      RIN = RINRTO * RINL
      YMCHIN = SQRT( ( ( HRGG - 1. ) * XMSQR + 2. ) /
     .                ( 2. * HRGG * XMSQR - ( HRGG - 1. ) ) )
      PRINT*,HRGG,RIN,PIN,YMCHIN
      PRINT*,HRGG,RINL,PINL,XMCHIN
      UVIN = XMCHIN * SQRT( HRGG * PINL / RINL ) -
     .       YMCHIN * SQRT( HRGG * PIN / RIN )
```

```
           END IF
           DO 175 IV = 1 , NV
           HYDVVV( IV , 1 ) = RINL
           HYDVVV( IV , 2 ) = 0.
           HYDVVV( IV , 3 ) = 0.
           HYDVVV( IV , 4 ) = PINL / HRGM
           HYDVVV( IV , 5 ) = HRGG
           XSS = XV( 1 , IV )
           IF( XSS . LT . -.0 ) THEN
           HYDVVV( IV , 1 ) = RIN
           HYDVVV( IV , 2 ) = UVIN * RIN
           HYDVVV( IV , 4 ) = PIN / HRGM + .5 * RIN * UVIN * UVIN
           END IF
   175     CONTINUE
           DO 170 IS = 1 , NS
           HYDV( IS , 1 ) = RINL
           HYDV( IS , 2 ) = 0.
           HYDV( IS , 3 ) = 0.
           HYDV( IS , 4 ) = PINL
           HYDV( IS , 5 ) = HRGG
           XSS = XS( 1 , IS )
           IF( XSS . LT . -.0 ) THEN
           HYDV( IS , 1 ) = RIN
           HYDV( IS , 2 ) = UVIN
           HYDV( IS , 4 ) = PIN
           END IF
   170     CONTINUE
C
           IF( IOPEOS . EQ . 1 ) THEN
           HRGGN = HRGG
           HRGGL = HRGG
           RINRTO = ( HRGGN + 1. ) * XMSQR /
     .              ( ( HRGGN - 1. ) * XMSQR + 2. )
           PINRTO = ( 2. * HRGGN * XMSQR - ( HRGGN - 1. ) ) /
     .                                     ( HRGGN + 1. )
           PIN = PINRTO * PINL
           RIN = RINRTO * RINL
           TTNN = PIN / ( HRGGN - 1. )
           RRNN = RIN
           TTNL = PINL / ( HRGGL - 1. )
           RRNL = RINL
           DO 1122 KI = 1 , 9
              CALL EOS( RRNN , TTNN , 1 , HRGGN )
              CALL EOS( RRNL , TTNL , 1 , HRGGL )
           RINRTO = ( HRGGN + 1. ) * XMSQR /
     .              ( ( HRGGN - 1. ) * XMSQR + 2. )
           PINRTO = ( 2. * HRGGN * XMSQR - ( HRGGN - 1. ) ) /
     .                                     ( HRGGN + 1. )
           RIN = RINRTO * RINL
           PIN = PINRTO * PINL
           TTNN = PIN / ( HRGGN - 1. )
           RRNN = RIN
           TTNL = PINL / ( HRGGL - 1. )
           RRNL = RINL
           YMCHIN = SQRT( ( ( HRGGN - 1. ) * XMSQR + 2. ) /
     .                    ( 2. * HRGGN * XMSQR - ( HRGGN - 1. ) ) )
           PRINT*,HRGGN,RIN,PIN,YMCHIN
           PRINT*,HRGGL,RINL,PINL,XMCHIN
  1122     CONTINUE
           UVIN = XMCHIN * SQRT( HRGGL * PINL / RINL ) -
     .            YMCHIN * SQRT( HRGGN * PIN / RIN )
           DO 172 IS = 1 , NS
           HYDV( IS , 5 ) = HRGGL
   172     CONTINUE
           END IF
           UIN = UVIN * COSS
           VIN = UVIN * SINN
C
           ENDIF
C(2)<<<<
C
           IF( MPRTCL . EQ . 1 ) THEN
             IKXY = 0
             DO 190 IKX = 1 , 30
```

```
            DO 190 IKY = 1 , 15
            IKXY = IKXY + 1
            XPRTCL( 1 , IKXY ) = ( IKX - 1 ) * .1 + .05
            XPRTCL( 2 , IKXY ) = ( IKY - 1 ) * .1 + .05
  190       CONTINUE
            NPT = IKXY
            PRINT *, NPT
            CALL PRLCTN
            PRINT *, NPT
          ENDIF
C
C=================================================================
C
C --- READ INPUT DATA FROM THE PREVIOUS RUN ----------------------------
C
           PRINT * ,ICOND,ICONP
           IF( ICONP . EQ . 1 ) THEN
             READ (88) RIN,PIN,RINL,PINL,UVIN,UIN,VIN,TT,
     .                   HYDMOM(1),HYDMOM(2),HYDMOM(4)
             PRINT *, RIN,PIN,UVIN,UIN,VIN,TT
             READ (88) ((HYDV(IS,IK),IK=1,5),IS=1,NS)
             READ (88) ((HYDVVV(IV,IK),IK=1,5),IV=1,NV)
             READ (88) IJKINT,(KSDELT(IS),IS=1,NS)
              IF( MPRTCL . EQ . 1 )
     .           READ (88) NPT,((XPRTCL(IK,IPT),IK=1,2),IPT=1,NPT),
     .                                   (IJKPRT(IPT),IPT=1,NPT)
          ENDIF
C
C=================================================================
C
C --- PERFORM THE ACTUAL CALCULATION --------------------------------
C
          CALL HYDRMN
C
C=================================================================
C
C --- EXIT POINT FROM PROGRAM ---------------------------------------
C
C         --------
          STOP 777
C         --------
C
C --- FORMATS -------------------------------------------------------
C
  101     FORMAT(1H ,'ICOND=',I2,5X,'ICONP=',I2,5X,'ITRIGR=',I2,5X,
     .              'IOPTN=',I2,/,1X,
     .              'XMCHIN=',F13.6,5X,'RIN=',F13.6,5X,'PIN=',F13.6,/,1X,
     .              'ALFA=',F13.6,5X,'HRGG=',F13.6,5X,'IHRN=',I2,5X,/,1X,
     .              'NTIME=',I2,5X,'MDUMP=',I5,5X,'NDUMP=',I5,5X,/,1X
     .              'KDUMP=',I5,5X,'IOSPCL=',I2,5X,'IOPLFT=',I2,5X,/,1X,
     .              'IOPRCN=',I2,5X,'IOPORD=',I2,5X,'IOPBYN=',I2,5X,/,1X
     .              'IAXSYM=',I2,5X,'IOPEOS=',I2,5X,'MPRTCL=',I6,5X,/,1X,
     .              'IOPINT=',I2,5X,'IOPADD=',I2,5X,'IOPDEL=',I2,5X,/,1X,
     .              'AREADD=',F13.6,5X,'AREDEL=',F13.6,5X,/,1X,
     .              'IWINDW=',I2,5X,'ISTATC=',I2)
C
C         ---
          END
```

```
      SUBROUTINE HYDRFL
C
C-------------------------------------------------------------I
C                                                             I
C      HYDRFL IS A 2 DIMENSIONAL RIEMANN SOLVER THAT COMPUTES THE I
C             FLUXES ACROSS NORMAL INTERFACES FOR UPDATING  SIDE  I
C             OR TRIANGLE BASED QUANTITIES.                   I
C                                                             I
C-------------------------------------------------------------I
C
      include     'cmsh00.h'
      include     'chyd00.h'
      include     'cint00.h'
      include     'cphs10.h'
      include     'cphs20.h'
C
C=====================================================================
C
      REAL DELP(MBP),WSOP(MBP),WSOM(MBP),WSOO(MBP),
     .     RSTAR(MBP),CSTAR(MBP),PMAX(MBP),PMIN(MBP)
      REAL RRIGHT(MBP),URIGHT(MBP),VRIGHT(MBP),PRIGHT(MBP)
      REAL RLEFTT(MBP),ULEFTT(MBP),VLEFTT(MBP),PLEFTT(MBP)
C
C=====================================================================
C
C
C --- BEGIN LOOP OVER ALL EDGES IN THE DOMAIN -------------------------
C
      DO 280 IH = 1 , 4
      DO 280 IS = 1 , NS
      HYDFLX( IS , IH ) = 0.
 280  CONTINUE
C
      NE1 = 1
      NE2 = NOFVEE( 1 )
      DO 110 INE = 1 , NVEEE
C
C --- FETCH HYDRO QUANTITIES -----------------------------------------
C     FOR LEFT AND RIGHT SIDE OF THE INTERFACE ON WHICH THE
C     RIEMANN PROBLEM IS SOLVED
C
      DO 120 IE = NE1 , NE2
          KE = IE - NE1 + 1
C
          RRR( KE ) = RR( IE )
          UUR( KE ) = UR( IE )
          VVR( KE ) = VR( IE )
          PPR( KE ) = PR( IE )
C
          RRL( KE ) = RL( IE )
          UUL( KE ) = UL( IE )
          VVL( KE ) = VL( IE )
          PPL( KE ) = PL( IE )
 120  CONTINUE
C
C --- ASSIGN GAMA A VALUE -------------------------------------------
C
      DO 130 KE = 1 , NOFVEE( INE )
         IE = KE + NE1 - 1
         ISL = JE( 3 , IE )
         ISR = JE( 4 , IE )
         GAMAL( KE ) = HYDV( ISL , 5 )
         IF( ISR . NE . 0 ) THEN
           GAMAR( KE ) = HYDV( ISR , 5 )
         ELSE
           GAMAR( KE ) = GAMAL( KE )
         END IF
C
C --- THIS SECTION OF CODE SOLVES FOR "PSTAR" AND "USTAR" IN -----------
C     THE RIEMANN PROBLEM USING NEWTON'S METHOD.
C
          WLEFT( KE ) = SQRT( GAMAL( KE ) * PPL( KE ) * RRL( KE ) )
          WRIGT( KE ) = SQRT( GAMAR( KE ) * PPR( KE ) * RRR( KE ) )
C
```

```
              WLESQ( KE ) = WLEFT( KE ) * WLEFT( KE )
              WRISQ( KE ) = WRIGT( KE ) * WRIGT( KE )
C
              PMIN( KE ) = AMIN1( PPL( KE ) , PPR( KE ) )
              PSML( KE ) = HRSM * PMIN( KE )
C
C --- FORM THE STARTING GUESS FOR THE SOLUTION ------------------------
C
              PSTAR( KE ) = ( WLEFT( KE ) * PPR( KE ) +
     .                        WRIGT( KE ) * PPL( KE ) -
     .                        WLEFT( KE ) * WRIGT( KE ) *
     .                      ( UUR( KE ) - UUL( KE ) ) ) /
     .                      ( WLEFT( KE ) + WRIGT( KE ) )
              PSTAR( KE ) = AMAX1( PSTAR( KE ) , PSML( KE ) )
  130   CONTINUE
C
              DO 140 I = 1 , IHRN
C
C --- BEGIN THE NEWTON ITERATION --------------------------------------
C
          DO 150 KE = 1 , NOFVEE( INE )
C
                CF = ( GAMAL( KE ) + 1 . ) / GAMAL( KE ) * .5
                WLEFS( KE ) = ( 1. + CF * ( PSTAR( KE ) /
     .                        PPL( KE ) - 1. ) ) * WLESQ( KE )
                WLEFT( KE ) = SQRT( WLEFS( KE ) )
                ZLEFT( KE ) = 2. * WLEFT( KE ) * WLEFS( KE ) /
     .                        ( WLESQ( KE ) + WLEFS( KE ) )
                USTL( KE )  = UUL( KE ) -
     .                      ( PSTAR( KE ) - PPL( KE ) ) / WLEFT( KE )
  150   CONTINUE
C
          DO 152 KE = 1 , NOFVEE( INE )
C
                CF = ( GAMAR( KE ) + 1 . ) / GAMAR( KE ) * .5
                WRIFS( KE ) = ( 1. + CF * ( PSTAR( KE ) /
     .                        PPR( KE ) - 1. ) ) * WRISQ( KE )
                WRIGT( KE ) = SQRT( WRIFS( KE ) )
                ZRIGT( KE ) = 2. * WRIGT( KE ) * WRIFS( KE ) /
     .                        ( WRISQ( KE ) + WRIFS( KE ) )
                USTR( KE )  = UUR( KE ) +
     .                      ( PSTAR( KE ) - PPR( KE ) ) / WRIGT( KE )
  152   CONTINUE
C
          DO 160 KE = 1 , NOFVEE( INE )
                DPST( KE )  = ZLEFT( KE ) * ZRIGT( KE ) *
     .                      ( USTR( KE ) - USTL( KE ) ) /
     .                      ( ZLEFT( KE ) + ZRIGT( KE ) )
                PSTAR( KE ) = PSTAR( KE ) - DPST( KE )
                PSTAR( KE ) = AMAX1( PSTAR( KE ) , PSML( KE ) )
  160       CONTINUE
  140       CONTINUE
C
C --- FORM FINAL SOLUTIONS ---------------------------------------------
C
          DO 170 KE = 1 , NOFVEE( INE )
C
              CF = ( GAMAL( KE ) + 1. ) / GAMAL( KE ) * .5
              WLEFT( KE ) = SQRT( WLESQ( KE ) * ( 1. +
     .                      CF * ( PSTAR( KE ) / PPL( KE ) - 1. ) ) )
  170   CONTINUE
C
          DO 172 KE = 1 , NOFVEE( INE )
C
              CF = ( GAMAR( KE ) + 1. ) / GAMAR( KE ) * .5
              WRIGT( KE ) = SQRT( WRISQ( KE ) * ( 1. +
     .                      CF * ( PSTAR( KE ) / PPR( KE ) - 1. ) ) )
  172   CONTINUE
C
          DO 180 KE = 1 , NOFVEE( INE )
              USTAR( KE ) = ( PPL( KE ) - PPR( KE ) +
     .                        WLEFT( KE ) * UUL( KE ) +
     .                        WRIGT( KE ) * UUR( KE ) ) /
     .                      ( WLEFT( KE ) + WRIGT( KE ) )
```

```
  180 CONTINUE
C
C --- BEGIN PROCEDURE TO OBTAIN FLUXES FROM REIMANN FORMALISM ----------
C
      DO 190 KE = 1 , NOFVEE( INE )
          IF( USTAR( KE ) . LE . 0.0 ) THEN
C
            RO( KE ) = RRR( KE )
            PO( KE ) = PPR( KE )
            UO( KE ) = UUR( KE )
            CO( KE ) = SQRT( GAMAR( KE ) * PPR( KE ) / RRR( KE ) )
            WO( KE ) = WRIGT( KE )
            ISN( KE ) = 1
C
            VGDNV( KE ) = VVR( KE )
C
          ELSE
C
            RO( KE ) = RRL( KE )
            PO( KE ) = PPL( KE )
            UO( KE ) = UUL( KE )
            CO( KE ) = SQRT( GAMAL( KE ) * PPL( KE ) / RRL( KE ) )
            WO( KE ) = WLEFT( KE )
            ISN( KE ) = - 1
C
            VGDNV( KE ) = VVL( KE )
          END IF
  190 CONTINUE
C
      DO 200 KE = 1 , NOFVEE( INE )
          DELP( KE ) = PSTAR( KE ) - PO( KE )
          WSOP( KE ) = ISN( KE ) * UO( KE ) + WO( KE ) / RO( KE )
          WSOM( KE ) = ISN( KE ) * UO( KE ) + CO( KE )
  200  CONTINUE
C
      DO 210 KE = 1 , NOFVEE( INE )
        IF( DELP( KE ) . GT . 0. ) THEN
          WSOO( KE ) = WSOP( KE )
        ELSE
          WSOO( KE ) = WSOM( KE )
        END IF
  210 CONTINUE
C
C --- USE OUTER STATE SOLUTION ----------------------------------------
C
      DO 220 KE = 1 , NOFVEE( INE )
          PGDNV( KE ) = PO( KE )
          UGDNV( KE ) = UO( KE )
          CGDNV( KE ) = CO( KE )
          RGDNV( KE ) = RO( KE )
  220 CONTINUE
C
C --- COMPUTE STARRED VALUES ------------------------------------------
C
      DO 230 KE = 1 , NOFVEE( INE )
         IE = KE + NE1 - 1
         ISL = JE( 3 , IE )
         ISR = JE( 4 , IE )
         IF( ISR . NE . 0 ) THEN
         GAMAG( KE ) = .5 * ( HYDV( ISL , 5 ) + HYDV( ISR , 5 ) )
         ELSE
         GAMAG( KE ) = HYDV( ISL , 5 )
         END IF
C
          RSTAR( KE ) = 1. / ( 1. / RO( KE ) - DELP( KE ) /
     .                    ( WO( KE ) * WO( KE ) ) )
C
          CSTAR( KE ) = SQRT( GAMAG( KE ) * PSTAR( KE ) / RSTAR( KE ) )
          WSOM( KE )  = ISN( KE ) * USTAR( KE ) + CSTAR( KE )
  230 CONTINUE
C
      DO 240 KE = 1 , NOFVEE( INE )
        IF( DELP( KE ) . GT . 0. ) THEN
          SPIN( KE ) = WSOP( KE )
```

```
            ELSE
              SPIN( KE ) = WSOM( KE )
            END IF
  240   CONTINUE
C
        DO 250 KE = 1 , NOFVEE( INE )
C
        IF( WSOO( KE ) . GE . 0. ) THEN
            IF( SPIN( KE ) . GE . 0. ) THEN
C
C --- USE THE STARRED STATE RESULTS -----------------------------------
C
                        RGDNV( KE ) = RSTAR( KE )
                        UGDNV( KE ) = USTAR( KE )
                        CGDNV( KE ) = CSTAR( KE )
                        PGDNV( KE ) = PSTAR( KE )
            ELSE
C
C --- EVALUATE THE INSIDE RAREFACTION WAVE ----------------------------
C
            HRGG = GAMAG( KE )
            HRGM = GAMAG( KE ) - 1.
            HRGP = GAMAG( KE ) + 1.
            CGDNV( KE ) = ( CSTAR( KE ) * 2. -
     .                    ISN( KE ) * USTAR( KE ) * HRGM ) / HRGP
            UGDNV( KE ) = - ISN( KE ) * CGDNV( KE )
            RGDNV( KE ) = ( CGDNV( KE ) / CO( KE ) ) **
     .                  ( 2. / HRGM ) * RO( KE )
            PGDNV( KE ) = CGDNV( KE ) * CGDNV( KE ) * RGDNV( KE ) / HRGG
C
            END IF
C
        END IF
  250   CONTINUE
C
            DO 142 IE = NE1 , NE2
            KE = IE - NE1 + 1
C
            RRR( KE ) = XN( IE )
            UUR( KE ) = YN( IE )
            VVR( KE ) = XXN( IE )
            PPR( KE ) = YYN( IE )
            PPL( KE ) = XE( 2 , IE )
            RRL( KE ) = XE( 1 , IE )
            UUL( KE ) = XYMIDL( IE )
C
  142   CONTINUE
C
C --- SEARCH FOR MINIMUM VALUE OF TIMESTEP ...DTT... ------------------
C
        DO 260 KE = 1 , NOFVEE( INE )
        CTT = SQRT( GAMAG( KE ) * PGDNV( KE ) / RGDNV( KE ) )
        VEL = UGDNV( KE )
C
        PROJCT = RRR( KE ) * VVR( KE ) + UUR( KE ) * PPR( KE )
        DTU = PPL( KE ) * ABS( PROJCT ) / ( CTT + ABS( VEL ) )
        DT1 = DTU * UUL( KE )
        DT2 = DTU - DT1
        DTT = AMIN1( DTT , DT1 , DT2 )
  260   CONTINUE
C
C --- NOW FIND THE FLUXES AT EACH INTERFACE ---------------------------
C
        DO 270 KE = 1 , NOFVEE( INE )
            HRGG = GAMAG( KE )
            HRGM = GAMAG( KE ) - 1.
            HRGP = GAMAG( KE ) + 1.
C
C ... FLUX FOR DENSITY ..............................................
C
        RO( KE ) = RGDNV( KE ) * UGDNV( KE )
C
C ... FLUX FOR MOMENTUM DENSITY .....................................
C
```

```
      UO( KE ) = PGDNV( KE ) * RRR( KE ) +
     .           RO( KE ) * ( UGDNV( KE ) * RRR( KE ) -
     .                        VGDNV( KE ) * UUR( KE ) )
      WO( KE ) = PGDNV( KE ) * UUR( KE ) +
     .           RO( KE ) * ( UGDNV( KE ) * UUR( KE ) +
     .                        VGDNV( KE ) * RRR( KE ) )
C
C ... FLUX FOR ENERGY DENSITY .........................................
C
      PO( KE ) = UGDNV( KE ) * ( PGDNV( KE ) * HRGG / HRGM +
     .     .5 * RGDNV( KE ) * ( UGDNV( KE ) * UGDNV( KE ) +
     .                          VGDNV( KE ) * VGDNV( KE ) ) )
C
  270 CONTINUE
C
C --- COLLECT INTERFACE FLUXES FOR EACH TRIANGLE ----------------------
C
      DO 290 IE = NE1 , NE2
       KE = IE - NE1 + 1
C
       ISL = JE( 3 , IE )
       ISR = JE( 4 , IE )
C
       DFLUX = RRL( KE )
C
      IF( JE( 5 , IE ) . EQ . 0 ) THEN
C
C ... FLUX FOR DENSITY ..............................................
C
      HYDFLX( ISL , 1 ) = HYDFLX( ISL , 1 ) + DFLUX * RO( KE )
      HYDFLX( ISR , 1 ) = HYDFLX( ISR , 1 ) - DFLUX * RO( KE )
C
C ... FLUX FOR MOMENTUM DENSITY ( U DIRECTION ) ......................
C
      HYDFLX( ISL , 2 ) = HYDFLX( ISL , 2 ) + DFLUX * UO( KE )
      HYDFLX( ISR , 2 ) = HYDFLX( ISR , 2 ) - DFLUX * UO( KE )
C
C ... FLUX FOR MOMENTUM DENSITY ( V DIRECTION ) ......................
C
      HYDFLX( ISL , 3 ) = HYDFLX( ISL , 3 ) + DFLUX * WO( KE )
      HYDFLX( ISR , 3 ) = HYDFLX( ISR , 3 ) - DFLUX * WO( KE )
C
C ... FLUX FOR ENERGY DENSITY ......................................
C
      HYDFLX( ISL , 4 ) = HYDFLX( ISL , 4 ) + DFLUX * PO( KE )
      HYDFLX( ISR , 4 ) = HYDFLX( ISR , 4 ) - DFLUX * PO( KE )
C
      ELSE
C
C ... FLUX FOR DENSITY ..............................................
C
      HYDFLX( ISL , 1 ) = HYDFLX( ISL , 1 ) + DFLUX * RO( KE )
C
C ... FLUX FOR MOMENTUM DENSITY ( U DIRECTION ) ......................
C
      HYDFLX( ISL , 2 ) = HYDFLX( ISL , 2 ) + DFLUX * UO( KE )
C
C ... FLUX FOR MOMENTUM DENSITY ( V DIRECTION ) ......................
C
      HYDFLX( ISL , 3 ) = HYDFLX( ISL , 3 ) + DFLUX * WO( KE )
C
C ... FLUX FOR ENERGY DENSITY ......................................
C
      HYDFLX( ISL , 4 ) = HYDFLX( ISL , 4 ) + DFLUX * PO( KE )
C
      END IF
  290 CONTINUE
C
      NE1 = NE2 + 1
      NE2 = NE2 + NOFVEE( INE + 1 )
  110 CONTINUE
C
C====================================================================
C
```

```
C --- EXIT POINT FROM SUBROUTINE ---------------------------------------
C
C     ------
      RETURN
C     ------
C
C     ---
      END
```

Thu Jul 1 14:15:40 1993 mainhd.f SUBROUTINE HYDRMN

```
      SUBROUTINE HYDRMN
C
C-------------------------------------------------------------I
C                                                             I
C      HYDRMN IS THE MAIN SUBROUTINE FOR THE UNSTRUCTURED GRID I
C             HYDRODYNAMIC SOLVER. THIS SUBROUTINE OBTAINS THE I
C             EDGE BASED FLUXES FOR EACH TRIANGLE/SIDE FROM    I
C             SUBROUTINE --- HYDRFL --- . IT ALSO CONTROLS     I
C             THE REFINEMENT AND COARSENING OF THE GRID.       I
C             THE SUBROUTINE GENERATES THE OUTPUT THAT IS USED I
C             FOR POST-PROCESSING.                             I
C                                                             I
C                                                             I
C-------------------------------------------------------------I
C
      include     'cmsh00.h'
      include     'chyd00.h'
      include     'cint00.h'
      include     'cphs10.h'
      include     'cphs20.h'
C
C======================================================================
C
      REAL RRN(MBP),URN(MBP),VRN(MBP),EPN(MBP),XSAR(MBP),
     .     TTN(MBP),XYRAD(MBP)
      INTEGER IEDIST(2)
C
C======================================================================
C
      CFL = 0.90
C
C --- SET SPECIFIC TIME FOR A DUMP -------------------------------------
C
      TLIMIT=30.
      FLATDR = .9
      LDUMP = KDUMP
      IF( IJKINT . EQ . 3 ) THEN
      LDUMP = 6
      IF( LDUMP . LT . KDUMP ) LDUMP = KDUMP
      END IF
C
      DO 120 JT  = 1 , NTIME
      DO 130 IT  = 1 , MDUMP
C
      DO 140 ITT = 1 , NDUMP
      IJKKJI = ( JT - 1 ) * NDUMP * MDUMP + ( IT - 1 ) * NDUMP + ITT
      IJKIJK = IJKINT + IJKKJI
C
      DO 142 IKT = 1 , LDUMP
C
C --- SELECT ORDER OF INTEGRATION --------------------------------------
C
      IF(IOPORD.EQ.1)THEN
         CALL FIRST
      ELSEIF(IOPORD.EQ.2)THEN
         CALL GRADNG
      ENDIF
C
C --- SET TIMESTEP TO HIGH VALUE IT WILL BE CALCULATED PROPERLY --------
C     IN THE FLUX SUBROUTINE
C
      DTT = 1.E24
C
```

```
C --- FIND THE FLUXES -----------------------------------------------
C
      CALL HYDRFL
C
      DTT = DTT * CFL
      TT = TT + DTT
      PRINT *,JT,IT,ITT,IKT,DTT,TT,NS
C
C --- INITIALIZE THE VERTEX BASED QUANTITIES NEEDED FOR COARSENING AND -
C     FOR REFINEMENT, AND FOR POST-PROCESSING
C
       DO 210 IV = 1 , NV
       PR( IV ) = 0.
       DO 210 IR = 1 , MHQ
       HYDVVV( IV , IR ) = 0.
210    CONTINUE
C
      NS1 = 1
      NS2 = NOFVES( 1 )
      DO 110 INS = 1 , NVEES
C
      DO 150 IS = NS1 , NS2
          KS = IS - NS1 + 1
          RRR( KS ) = HYDV( IS , 1 )
          UUR( KS ) = HYDV( IS , 2 )
          VVR( KS ) = HYDV( IS , 3 )
          PPR( KS ) = HYDV( IS , 4 )
C
          RRL( KS ) = HYDFLX( IS , 1 )
          UUL( KS ) = HYDFLX( IS , 2 )
          VVL( KS ) = HYDFLX( IS , 3 )
          PPL( KS ) = HYDFLX( IS , 4 )
C
          XSAR( KS ) = SAREA( IS )
150    CONTINUE
C
       DO 170 KS = 1 , NOFVES( INS )
          IS = KS + NS1 - 1
         GAMAG( KS ) = HYDV( IS , 5 )
         HRGM = GAMAG( KS ) - 1.
C
       RRN( KS ) = RRR( KS )
       URN( KS ) = RRR( KS ) * UUR( KS )
       VRN( KS ) = RRR( KS ) * VVR( KS )
       EPN( KS ) = PPR( KS ) / HRGM + .5 * RRR( KS ) *
     .                        ( UUR( KS ) * UUR( KS ) +
     .                          VVR( KS ) * VVR( KS ) )
170    CONTINUE
C
C===================================================================
C
C --- COMPUTING THE SOURCE TERM ASSOCIATED WITH AXI-SYMMETRIC CASE -----
C
          XYDUMY = 1. / 6.283185307
          DO 188 KS = 1 , NOFVES( INS )
          XYRAD( KS ) = XYDUMY
188       CONTINUE
C
C --- Y-AXIS IS AXIS OF SYMMETRY -------------------------------------
C
          IF( IAXSYM . EQ . 2 )THEN
          DO 180 KS = 1 , NOFVES( INS )
          IS = KS + NS1 - 1
          XS2S = XS( 1 , IS )
          XYRAD( KS ) = XS2S
          IF( XS2S . GT . .0005 ) THEN
            DTA = DTT * UUR( KS ) / XS2S
            RRN( KS ) = RRN( KS ) * ( 1. - DTA )
            URN( KS ) = URN( KS ) * ( 1. - DTA )
            VRN( KS ) = VRN( KS ) * ( 1. - DTA )
            EPN( KS ) = EPN( KS ) * ( 1. - DTA ) - PPR( KS ) * DTA
          END IF
180       CONTINUE
C
```

```
C --- X-AXIS IS AXIS OF SYMMETRY -------------------------------------
C
           ELSEIF( IAXSYM . EQ . 1 )THEN
           DO 182 KS = 1 , NOFVES( INS )
           IS = KS + NS1 - 1
           XS2S = XS( 2 , IS )
           XYRAD( KS ) = XS2S
           IF( XS2S . GT . .0005 ) THEN
             DTA = DTT * VVR( KS ) / XS2S
             RRN( KS ) = RRN( KS ) * ( 1. - DTA )
             URN( KS ) = URN( KS ) * ( 1. - DTA )
             VRN( KS ) = VRN( KS ) * ( 1. - DTA )
             EPN( KS ) = EPN( KS ) * ( 1. - DTA ) - PPR( KS ) * DTA
           END IF
  182   CONTINUE
          ENDIF
C
C --- COMPUTE THE EFFECT OF THE BOUYANCY(GRAVITY) TERM ----------------
C
          GRAVTY = 9.81
C
          IF( IOPBYN . EQ . 2 )THEN
            DO 184 KS = 1 , NOFVES( INS )
            DTA = DTT * RRR( KS ) * GRAVTY
              VRN( KS ) = VRN( KS ) - DTA
              EPN( KS ) = EPN( KS ) - DTA * VVR( KS )
  184      CONTINUE
C
          ELSEIF( IOPBYN . EQ . 1 )THEN
            DO 186 KS = 1 , NOFVES( INS )
            DTA = DTT * RRR( KS ) * GRAVTY
              URN( KS ) = URN( KS ) - DTA
              EPN( KS ) = EPN( KS ) - DTA * UUR( KS )
  186      CONTINUE
            END IF
C
C====================================================================
C
C --- UPDATE THE HYDRODYNAMIC QUANTITIES ------------------------------
C     STORING THE FLUXES FOR THE REFINEMENT/COARSENING STEPS
C
        DO 190 KS = 1 , NOFVES( INS )
        IS = KS + NS1 - 1
        DTA = DTT * XSAR( KS )
C
        RRLL = RRL( KS )
        UULL = UUL( KS )
        VVLL = VVL( KS )
        RRN( KS ) = RRN( KS ) - RRLL * DTA
        URN( KS ) = URN( KS ) - UULL * DTA
        VRN( KS ) = VRN( KS ) - VVLL * DTA
C
        PPLL = PPL( KS )
        HYDFLX( IS , 4 ) = ABS( PPLL ) / EPN( KS ) * DTA
        EPN( KS ) = EPN( KS ) - PPLL * DTA
C
  190   CONTINUE
C
        DO 202 IS = NS1 , NS2
        KS = IS - NS1 + 1
        ENERGY = 1. / RRN( KS ) * ( URN( KS ) * URN( KS ) +
     .                              VRN( KS ) * VRN( KS ) )
        TTN( KS ) = EPN( KS ) - .5 * ENERGY
        HYDFLX( IS , 1 ) = ENERGY / TTN( KS )
        HYDFLX( IS , 2 ) = RRN( KS )
C
  202   CONTINUE
C --- EQUATION OF STATE FOR AIR ---------------------------------------
C
        IF( IOPEOS . EQ . 1 )THEN
          CALL EOS( RRN , TTN , NOFVES( INS ) , GAMAG )
        ELSE
        ENDIF
C
```

```
C --- ACCUMULATE VALUES AT THE VERTICES FOR ADAPTATION AND ALSO --------
C     FOR POST-PROCESSING
C
          DO 220 KS = 1 , NOFVES( INS )
          IS = KS + NS1 - 1
C
          IV1 = JS( 1 , IS )
          IV2 = JS( 2 , IS )
          IV3 = JS( 3 , IS )
C
          VOLUME = 6.283185307 * XYRAD( KS )
C
          XYAREA = XS( 3 , IS ) * VOLUME
          XYFDR = XYAREA * RRN( KS )
          XYFDU = XYAREA * URN( KS )
          XYFDV = XYAREA * VRN( KS )
          XYFDP = XYAREA * EPN( KS )
          XYFDG = XYAREA * GAMAG( KS )
C
          HYDVVV( IV1 , 1 ) = HYDVVV( IV1 , 1 ) + XYFDR
          HYDVVV( IV1 , 2 ) = HYDVVV( IV1 , 2 ) + XYFDU
          HYDVVV( IV1 , 3 ) = HYDVVV( IV1 , 3 ) + XYFDV
          HYDVVV( IV1 , 4 ) = HYDVVV( IV1 , 4 ) + XYFDP
          HYDVVV( IV1 , 5 ) = HYDVVV( IV1 , 5 ) + XYFDG
          PR( IV1 ) = PR( IV1 ) + XYAREA
C
          HYDVVV( IV2 , 1 ) = HYDVVV( IV2 , 1 ) + XYFDR
          HYDVVV( IV2 , 2 ) = HYDVVV( IV2 , 2 ) + XYFDU
          HYDVVV( IV2 , 3 ) = HYDVVV( IV2 , 3 ) + XYFDV
          HYDVVV( IV2 , 4 ) = HYDVVV( IV2 , 4 ) + XYFDP
          HYDVVV( IV2 , 5 ) = HYDVVV( IV2 , 5 ) + XYFDG
          PR( IV2 ) = PR( IV2 ) + XYAREA
C
          HYDVVV( IV3 , 1 ) = HYDVVV( IV3 , 1 ) + XYFDR
          HYDVVV( IV3 , 2 ) = HYDVVV( IV3 , 2 ) + XYFDU
          HYDVVV( IV3 , 3 ) = HYDVVV( IV3 , 3 ) + XYFDV
          HYDVVV( IV3 , 4 ) = HYDVVV( IV3 , 4 ) + XYFDP
          HYDVVV( IV3 , 5 ) = HYDVVV( IV3 , 5 ) + XYFDG
          PR( IV3 ) = PR( IV3 ) + XYAREA
C
          IENUMR = 0
          IE1 = IABS( JS( 4 , IS ) )
          IJE5 = JE( 5 , IE1 )
          IF( IJE5 . NE . 0 ) THEN
          IENUMR = IENUMR + 1
          IEDIST( IENUMR ) = IE1
          END IF
          IE2 = IABS( JS( 5 , IS ) )
          IJE5 = JE( 5 , IE2 )
          IF( IJE5 . NE . 0 ) THEN
          IENUMR = IENUMR + 1
          IEDIST( IENUMR ) = IE2
          END IF
          IE3 = IABS( JS( 6 , IS ) )
          IJE5 = JE( 5 , IE3 )
          IF( IJE5 . NE . 0 ) THEN
          IENUMR = IENUMR + 1
          IEDIST( IENUMR ) = IE3
          END IF
C
          IF( IENUMR . NE . 0 ) THEN
          DO 322 IK = 1 , IENUMR
          IEK = IEDIST( IENUMR )
          IJE55 = JE( 5 , IEK )
          RRNN = RRN( KS )
          URNN = URN( KS )
          VRNN = VRN( KS )
          EPNN = EPN( KS )
C          IF( IJE55 . EQ . 6 . OR . IJE55 . EQ . 5 ) THEN
C          UUVV =    - ( URN( KS ) * XN( IEK ) +
C     .                  VRN( KS ) * YN( IEK ) )
C          VVUU =      - URN( KS ) * YN( IEK ) +
C     .                  VRN( KS ) * XN( IEK )
C          URNN = UUVV * XN( IEK ) - VVUU * YN( IEK )
```

```
C            VRNN = UUVV * YN( IEK ) + VVUU * XN( IEK )
C            ELSE IF( IJE55 . EQ . 8 ) THEN
C            RRNN = RIN
C            URNN = RIN * UIN
C            VRNN = RIN * VIN
C            EPNN = PIN / HRGM + .5 * RIN * UVIN * UVIN
C            END IF
C
           XYFDR = XYAREA * RRNN
           XYFDU = XYAREA * URNN
           XYFDV = XYAREA * VRNN
           XYFDP = XYAREA * EPNN
           XYFDG = XYAREA * GAMAG( KS )
C
           IV1 = JE( 1 , IEK )
           IV2 = JE( 2 , IEK )
           HYDVVV( IV1 , 1 ) = HYDVVV( IV1 , 1 ) + XYFDR
           HYDVVV( IV1 , 2 ) = HYDVVV( IV1 , 2 ) + XYFDU
           HYDVVV( IV1 , 3 ) = HYDVVV( IV1 , 3 ) + XYFDV
           HYDVVV( IV1 , 4 ) = HYDVVV( IV1 , 4 ) + XYFDP
           HYDVVV( IV1 , 5 ) = HYDVVV( IV1 , 5 ) + XYFDG
           PR( IV1 ) = PR( IV1 ) + XYAREA
C
           HYDVVV( IV2 , 1 ) = HYDVVV( IV2 , 1 ) + XYFDR
           HYDVVV( IV2 , 2 ) = HYDVVV( IV2 , 2 ) + XYFDU
           HYDVVV( IV2 , 3 ) = HYDVVV( IV2 , 3 ) + XYFDV
           HYDVVV( IV2 , 4 ) = HYDVVV( IV2 , 4 ) + XYFDP
           HYDVVV( IV2 , 5 ) = HYDVVV( IV2 , 5 ) + XYFDG
           PR( IV2 ) = PR( IV2 ) + XYAREA
  322      CONTINUE
           END IF
C
  220      CONTINUE
C
C --- CONSTRUCT NONCONSERVED HYDRODYNAMIC QUATITIES --------------------
C
         DO 195 IS = NS1 , NS2
         KS = IS - NS1 + 1
         HDUM            = 1. / RRN( KS )
         HYDV( IS , 1 ) = RRN( KS )
         HYDV( IS , 2 ) = URN( KS ) * HDUM
         HYDV( IS , 3 ) = VRN( KS ) * HDUM
         HYDV( IS , 5 ) = GAMAG( KS )
         HYDV( IS , 4 ) = TTN( KS ) * ( HYDV( IS , 5 ) - 1. )
  195    CONTINUE
C
         NS1 = NS2 + 1
         NS2 = NS2 + NOFVES( INS + 1 )
  110    CONTINUE
C
C --- END OF LOOP OVER TRIANGLES ---------------------------------------
C
C======================================================================
C
C --- CALL FOR PARTICLE TRACERS ----------------------------------------
C
        IF( MPRTCL . EQ . 1 )THEN
C
          CALL PRPTHC
C
        ENDIF
C
C --- END OF INNER LOOP OVER ... KDUMP ... -----------------------------
C
C======================================================================
C
C --- NORMALIZE CONSERVATIVE  VERTEX BASED QUANTITIES ------------------
C
        DO 230 IV = 1 , NV
        VAREA = 1. / PR( IV )
        DO 230 IR = 1 , MHQ
        HYDVVV( IV , IR ) = HYDVVV( IV , IR ) * VAREA
  230   CONTINUE
C
```

```
  142   CONTINUE
C --- WRITE THE DUMP FILE DATA FOR POST-PROCESSING --------------------
C
        IF( IT . EQ . MDUMP . AND . ITT . EQ . NDUMP ) THEN
          WRITE (9) NV,NE,NS,NPT,NTIME
          WRITE (9) ((XV(IK,IV),IK=1,2),IV=1,NV)
          WRITE (9) (JV(2,IV),IV=1,NV)
          WRITE (9) ((JE(KK,IE),KK=1,5),IE=1,NE)
          WRITE (9) ((JS(KK,IS),KK=1,6),IS=1,NS)
          WRITE (9) ((XS(KK,IS),KK=1,2),IS=1,NS)
          WRITE (9) RIN,PIN,UVIN,UIN,VIN,TT,IOPLFT
          WRITE (9) ((HYDV(IS,IK),IK=1,5),IS=1,NS)
C
C --- WRITE OUT PARTICLE TRACER DATA ---------------------------------
C
          IF( MPRTCL . EQ . 1 )THEN
            WRITE (9) ((XPRTCL(IK,IPT),IK=1,2),IPT=1,NPT),
     .                  ((WPRTCL(IK,IPT),IK=1,2),IPT=1,NPT)
          ENDIF
C
C --- PRINT CONSOLE MESSAGE AT END OF LOOP ----------------------------
C
          PRINT * , JT,NV,NE,NS
C
        END IF
C
C=====================================================================
C
C               I--------------------------------I
C               I  REFINEMENT/ADDITION OF POINTS  I
C               I--------------------------------I
C
C
C --- CALCULATE THE GRADIENT OF THE MACH NUMBER FOR STEADY STATE ------
C     ADAPTIVE STEP AND GENERATE THE QUANTITIES ON WHICH WE ADAPT.
C
C --- ADAPTATION TO STATIC QUANTITIES BASED ON GRADIENTS OF ------------
C     MACH NUMBER, PRESSURE, AND DENSITY. OVERRIDES
C     ADAPTATION ON DYNAMIC FLUXES OF ENERGY AND DENSITY
C
        IF( ISTATC . EQ . 1 ) THEN
        CALL GRDFLX
        DO 240 IS = 1 , NS
        HYDFLX( IS , 1 ) = ABS( PL( IS ) ) + ABS( PR( IS ) )
        HYDFLX( IS , 2 ) = ABS( RL( IS ) ) + ABS( RR( IS ) )
        HYDFLX( IS , 4 ) = ABS( VL( IS ) ) + ABS( VR( IS ) )
  240   CONTINUE
C
        ELSE
C
        CALL GRDENG
        DO 242 IS = 1 , NS
        HYDFLX( IS , 1 ) = ( UL( IS ) * UL( IS ) + UR( IS ) * UR( IS ) ) /
     .                     ( HYDFLX( IS , 1 ) + 1.E-12 )
        HYDFLX( IS , 2 ) = ( RL( IS ) * RL( IS ) + RR( IS ) * RR( IS ) ) /
     .                     ( HYDFLX( IS , 2 ) + 1.E-12 )
        HYDFLX( IS , 4 ) = ( VL( IS ) * VL( IS ) + VR( IS ) * VR( IS ) ) /
     .                     ( HYDFLX( IS , 4 ) + 1.E-12 )
  242   CONTINUE
        END IF
C
        DYDMOM = HYDFLX( 1 , 4 )
        DO 250 IS = 1 , NS
        DYDMOM = AMAX1( DYDMOM , HYDFLX( IS , 4 ) )
  250   CONTINUE
        HYDMOM( 4 ) = .5 * ( DYDMOM + HYDMOM( 4 ) )
        PRINT*,HYDMOM( 4 )
C
        DYDMOM = HYDFLX( 1 , 2 )
        DO 260 IS = 1 , NS
        DYDMOM = AMAX1( DYDMOM , HYDFLX( IS , 2 ) )
  260   CONTINUE
        HYDMOM( 2 ) = .5 * ( DYDMOM + HYDMOM( 2 ) )
        PRINT*,HYDMOM( 2 )
```

```
C
      DYDMOM = HYDFLX( 1 , 1 )
      DO 270 IS = 1 , NS
      DYDMOM = AMAX1( DYDMOM , HYDFLX( IS , 1 ) )
  270 CONTINUE
      HYDMOM( 1 ) = .5 * ( DYDMOM + HYDMOM( 1 ) )
      PRINT*,HYDMOM( 1 )
C
C --- REFINEMENT STEP DONE HERE ---------------------------------------
C
      IF(IOPADD.EQ.1) THEN
        NOFDIV = 4
        CALL DYNPTN( AREADD , NOFDIV , IJKIJK , LTRIG )
        NOFDIV = 2
        CALL DYYPTN( AREADD , NOFDIV , IJKIJK , LTRIG )
        NOFDIV = 1
        CALL DYYPTN( AREADD , NOFDIV , IJKIJK , LTRIG )
        CALL DYYPTN( AREADD , NOFDIV , IJKIJK , LTRIG )
C
        PRINT*,NV,NE,NS
      ENDIF
  140 CONTINUE
C
C --- END OF OUTER LOOP DEFINED BY ...NDUMP... ------------------------
C
C=====================================================================
C
C            I--------------------------------I
C            I  COARSENING/DELETION OF POINTS  I
C            I--------------------------------I
C
C
       IF(IOPDEL.EQ.1)THEN
          IF( IJKIJK . GT . 19 ) CALL DELPTNT( AREDEL , IJKIJK )
         PRINT*,NV,NE,NS
       ENDIF
  130  CONTINUE
C
C --- END OF OUTERMOST LOOP DEFINED BY ...MDUMP... --------------------
C
C=====================================================================
C
C            I--------------------------------I
C            I  DIAGNOSTIC FOR LIFT/DRAG      I
C            I--------------------------------I
C
      IF(IOPLFT.EQ.1)THEN
        CALL LIFTDR
      ENDIF
C
C=====================================================================
C
C            I--------------------------------I
C            I  OUTPUT FILE FOR RESTARTS      I
C            I--------------------------------I
C
C
       REWIND 88
       ITERAT = ITERAT + 1
       WRITE (88) NV,NVMK,NE,NEMK,NS,NSMK,ITERAT
       WRITE (88) ((JV(KK,IV),KK=1,2),(XV(IK,IV),IK=1,2),IV=1,NV)
       WRITE (88) ((JE(KK,IE),KK=1,5),(XE(KI,IE),KI=1,2),IE=1,NE)
       WRITE (88) (XN(IE),YN(IE),XXN(IE),YYN(IE),IE=1,NE)
       WRITE (88) ((JS(KK,IS),KK=1,6),(XS(KI,IS),KI=1,3),IS=1,NS)
       WRITE (88) (XMIDL(IE),YMIDL(IE),XYMIDL(IE),IE=1,NE)
       WRITE (88) SAREVG,NVECE,NREME,NVECV,NREMV,NVECS,NREMS
       WRITE (88) RIN,PIN,RINL,PINL,UVIN,UIN,VIN,TT,
     .              HYDMOM(1),HYDMOM(2),HYDMOM(4)
       WRITE (88) ((HYDV(IS,IK),IK=1,5),IS=1,NS)
       WRITE (88) ((HYDVVV(IV,IK),IK=1,5),IV=1,NV)
       WRITE (88) IJKIJK,(KSDELT(IS),IS=1,NS)
        IF( MPRTCL . EQ . 1 )
     . WRITE (88) NPT,((XPRTCL(IK,IPT),IK=1,2),IPT=1,NPT),
     .                                (IJKPRT(IPT),IPT=1,NPT)
```

```
C
       REWIND 8
       WRITE (8) NV,NVMK,NE,NEMK,NS,NSMK,ITERAT
       WRITE (8) ((JV(KK,IV),KK=1,2),(XV(IK,IV),IK=1,2),IV=1,NV)
       WRITE (8) ((JE(KK,IE),KK=1,5),(XE(KI,IE),KI=1,2),IE=1,NE)
       WRITE (8) (XN(IE),YN(IE),XXN(IE),YYN(IE),IE=1,NE)
       WRITE (8) ((JS(KK,IS),KK=1,6),(XS(KI,IS),KI=1,3),IS=1,NS)
       WRITE (8) (XMIDL(IE),YMIDL(IE),XYMIDL(IE),IE=1,NE)
       WRITE (8) SAREVG,NVECE,NREME,NVECV,NREMV,NVECS,NREMS
       WRITE (8) RIN,PIN,RINL,PINL,UVIN,UIN,VIN,TT,
     .             HYDMOM(1),HYDMOM(2),HYDMOM(4)
       WRITE (8) ((HYDV(IS,IK),IK=1,5),IS=1,NS)
       WRITE (8) ((HYDVVV(IV,IK),IK=1,5),IV=1,NV)
       WRITE (8) IJKIJK,(KSDELT(IS),IS=1,NS)
        IF( MPRTCL . EQ . 1 )
     . WRITE (8) NPT,((XPRTCL(IK,IPT),IK=1,2),IPT=1,NPT),
     .                                (IJKPRT(IPT),IPT=1,NPT)
C
 120   CONTINUE
C
C --- END MAIN SEQUENCE LOOP DEFINED BY ...NTIME...--------------------
C
C===================================================================
C
C --- EXIT POINT FROM SUBROUTINE -------------------------------------
C
C     ------
      RETURN
C     ------
C
C --- FORMATS --------------------------------------------------------
C
C     ---
      END
```

Thu Jul 1 14:15:40 1993 mainhd.f SUBROUTINE GEOMTR

```
      SUBROUTINE GEOMTR
C
C-------------------------------------------------------------I
C                                                             I
C      GEOMTR COMPUTE GEOMETRICAL PARAMETERS TO COMPLETE THE GRID I
C             DEFINITION NEEDED BY THE CODE.                  I
C                                                             I
C                                                             I
C-------------------------------------------------------------I
C
      include     'cmsh00.h'
      include     'chyd00.h'
      include     'cint00.h'
      include     'cphs10.h'
      include     'cphs20.h'
C
C===================================================================
C
       REAL XELEFT(MBP),YELEFT(MBP),XERIGT(MBP),YERIGT(MBP)
C
C===================================================================
C
C
C --- MAKE SURE THAT THE DOMAIN IS ALWAYS TO THE LEFT OF BOUNDARY ------
C     EDGES BY ORIENTING THEM CORRECTLY.
C
      DO 105 IE = 1 , NE
        IJE5 = JE( 5 , IE )
        IF( IJE5 . NE . 0 ) THEN
          ISR = JE( 4 , IE )
          IF( ISR . NE . 0 ) THEN
            IV2 = JE( 1 , IE )
            IV1 = JE( 2 , IE )
            JE( 1 , IE ) = IV1
            JE( 2 , IE ) = IV2
            JE( 3 , IE ) = ISR
            JE( 4 , IE ) = 0
```

```
                DO 106 IR = 1 , 3
                JEE = IABS( JS( IR + 3 , ISR ) )
                IF( JEE . EQ . IE ) JS( IR + 3 , ISR ) = IE
  106           CONTINUE
              ENDIF
            ENDIF
  105   CONTINUE
C
C --- FIND UNIT VECTORS NORMAL TO THE EDGES ---------------------------
C       AND EDGE CROSSING OF LINE BETWEEN TRIANGLE BARICENTERS
C       AND ALSO TRIANGLE AREAS.
C
        NE1 = 1
        NE2 = NOFVEE( 1 )
        DO 110 INE = 1 , NVEEE
        DO 140 IE = NE1 , NE2
          KE = IE - NE1 + 1
          IV1 = JE( 1 , IE )
          IV2 = JE( 2 , IE )
          ISL = JE( 3 , IE )
          ISR = JE( 4 , IE )
C
C --- FIND UNIT VECTOR NORMAL TO AN EDGE ------------------------------
C       STORED IN XN(IE),YN(IE)
C
          DXD = XV( 1 , IV2 ) - XV( 1 , IV1 )
          DYD = XV( 2 , IV2 ) - XV( 2 , IV1 )
          XE( 1 , IE ) = SQRT( DXD * DXD + DYD * DYD )
          XEY = 1. / XE( 1 , IE )
          XD = DXD * XEY
          YD = DYD * XEY
C
          XN( IE ) =   YD
          YN( IE ) = - XD
C
          IJE5 = JE( 5 , IE )
C
C=====================================================================
C
C --- BOUNDARY TRIANGLES ----------------------------------------------
C
         IF( IJE5 . NE . 0 ) THEN
         IV3 = JS( 1 , ISL )
         IF( IV3 . EQ . IV1 . OR . IV3 . EQ . IV2 ) IV3 = JS( 2 , ISL )
         IF( IV3 . EQ . IV1 . OR . IV3 . EQ . IV2 ) IV3 = JS( 3 , ISL )
         XELEFT( KE ) = ( XV( 1 , IV3 ) + XV( 1 , IV2 ) +
     .                                    XV( 1 , IV1 ) ) * THIRD
         YELEFT( KE ) = ( XV( 2 , IV3 ) + XV( 2 , IV2 ) +
     .                                    XV( 2 , IV1 ) ) * THIRD
C
         AA = XV( 1 , IV2 ) - XV( 1 , IV1 )
         BB = XV( 2 , IV2 ) - XV( 2 , IV1 )
         CC = XELEFT( KE ) - XV( 1 , IV1 )
         DD = YELEFT( KE ) - XV( 2 , IV1 )
         EE = ( AA * CC + BB * DD ) * XEY * XEY
         XERIGT( KE ) = XV( 1 , IV1 ) + AA * EE
         YERIGT( KE ) = XV( 2 , IV1 ) + BB * EE
C
         DXD = XERIGT( KE ) - XELEFT( KE )
         DYD = YERIGT( KE ) - YELEFT( KE )
         XE( 2 , IE ) = SQRT( DXD * DXD + DYD * DYD )
C
C --- UNIT VECTOR FROM LEFT TO RIGHT BARI-CENTER AT INTERFACE ----------
C       STORED IN XXN(IE),YYN(IE)
C
         XY = 1. / XE( 2 , IE )
         XXN( IE ) = DXD * XY
         YYN( IE ) = DYD * XY
C
C --- LENGTH OF LINE BETWEEN BARI-CENTERS -----------------------------
C       STORED IN XE(2,IE)
C
         XE( 2 , IE ) = 2. * XE( 2 , IE )
C
```

```
C --- COORDINATES OF BARI-CENTERS FOR EACH TRIANGLE -------------------
C     STORED IN XS(1,IS),XS(2,IS)
C
       XS( 1 , ISL ) = XELEFT( KE )
       XS( 2 , ISL ) = YELEFT( KE )
C
C --- INTERSECTION POINT ON INTERFACE FOR LINE CONNECTING BARI-CENTERS -
C     STORED IN XMIDL(IE),YMIDL(IE) AND FRACTION OF LENGHT BETWEEN
C     LEFT BARI-CENTER TO INTERSECTION POINT IN XYMIDL(IE).
C
       XYMIDL( IE ) = .5
       XMIDL( IE ) = XERIGT( KE )
       YMIDL( IE ) = YERIGT( KE )
C
C=======================================================================
C
C
C --- REGULAR TRIANGLES ------------------------------------------------
C
       ELSE
       IV3 = JS( 1 , ISL )
       IF( IV3 . EQ . IV1 . OR . IV3 . EQ . IV2 ) IV3 = JS( 2 , ISL )
       IF( IV3 . EQ . IV1 . OR . IV3 . EQ . IV2 ) IV3 = JS( 3 , ISL )
       XELEFT( KE ) = ( XV( 1 , IV3 ) + XV( 1 , IV2 ) +
     .                                  XV( 1 , IV1 ) ) * THIRD
       YELEFT( KE ) = ( XV( 2 , IV3 ) + XV( 2 , IV2 ) +
     .                                  XV( 2 , IV1 ) ) * THIRD
C
       IV3 = JS( 1 , ISR )
       IF( IV3 . EQ . IV1 . OR . IV3 . EQ . IV2 ) IV3 = JS( 2 , ISR )
       IF( IV3 . EQ . IV1 . OR . IV3 . EQ . IV2 ) IV3 = JS( 3 , ISR )
       XERIGT( KE ) = ( XV( 1 , IV3 ) + XV( 1 , IV2 ) +
     .                                  XV( 1 , IV1 ) ) * THIRD
       YERIGT( KE ) = ( XV( 2 , IV3 ) + XV( 2 , IV2 ) +
     .                                  XV( 2 , IV1 ) ) * THIRD
C
        DXD = XERIGT( KE ) - XELEFT( KE )
        DYD = YERIGT( KE ) - YELEFT( KE )
C
C --- LENGTH OF LINE BETWEEN BARI-CENTERS ------------------------------
C     STORED IN XE(2,IE)
C
        XE( 2 , IE ) = SQRT( DXD * DXD + DYD * DYD )
C
C --- UNIT VECTOR FROM LEFT TO RIGHT BARI-CENTER AT INTERFACE ----------
C     STORED IN XXN(IE),YYN(IE)
C
        XY = 1. / XE( 2 , IE )
        XXN( IE ) = DXD * XY
        YYN( IE ) = DYD * XY
C
C --- COORDINATES OF BARI-CENTERS FOR EACH TRIANGLE -------------------
C     STORED IN XS(1,IS),XS(2,IS)
C
       XS( 1 , ISL ) = XELEFT( KE )
       XS( 2 , ISL ) = YELEFT( KE )
       XS( 1 , ISR ) = XERIGT( KE )
       XS( 2 , ISR ) = YERIGT( KE )
C
       AA = XV( 1 , IV2 ) - XV( 1 , IV1 )
       BB = XV( 2 , IV2 ) - XV( 2 , IV1 )
       CC = XELEFT( KE ) - XERIGT( KE )
       DD = YELEFT( KE ) - YERIGT( KE )
       ACA = XERIGT( KE ) - XV( 1 , IV1 )
       DBD = YERIGT( KE ) - XV( 2 , IV1 )
       EE = ( ACA * DD - DBD * CC ) / ( AA * DD - BB * CC )
C
C --- INTERSECTION POINT ON INTERFACE FOR LINE CONNECTING BARI-CENTERS -
C     STORED IN XMIDL(IE),YMIDL(IE) AND FRACTION OF LENGHT BETWEEN
C     LEFT BARI-CENTER TO INTERSECTION POINT IN XYMIDL(IE).
C
       XMIDL( IE ) = XV( 1 , IV1 ) + AA * EE
       YMIDL( IE ) = XV( 2 , IV1 ) + BB * EE
C
```

```
              XEMID = XMIDL( IE ) - XELEFT( KE )
              YEMID = YMIDL( IE ) - YELEFT( KE )
C
              XYMIDL( IE ) = SQRT( XEMID * XEMID + YEMID * YEMID ) * XY
C
              ENDIF
  140     CONTINUE
C
              NE1 = NE2 + 1
              NE2 = NE2 + NOFVEE( INE + 1 )
  110     CONTINUE
C
C --- CALCULATE AREA OF TRIANGLES ------------------------------------
C
              DO 150 IS = 1 , NS
              IV1 = JS( 1 , IS )
              IV2 = JS( 2 , IS )
              IV3 = JS( 3 , IS )
              DX  = XV( 1 , IV2 ) - XV( 1 , IV1 )
              DXX = XV( 1 , IV3 ) - XV( 1 , IV1 )
              DY  = XV( 2 , IV2 ) - XV( 2 , IV1 )
              DYY = XV( 2 , IV3 ) - XV( 2 , IV1 )
              XS( 3 , IS ) = .5 * ( DX * DYY - DXX * DY )
  150         CONTINUE
C
              PRINT * , NE,NS
C
C --- FIND AN EDGE ASSOCIATED WITH A VERTEX ----------------------------
C     THE VALUE WILL BE NEGATIVE IF ON THE BOUNDARY
C
          DO 180 IV = 1 , NV
          JV( 2 , IV ) = 0
  180     CONTINUE
C
          DO 160 IE = 1 , NE
          IV1 = JE( 1 , IE )
          IJE5 = JE( 5 , IE )
          IF( IJE5 . NE . 0 ) THEN
          JV( 2 , IV1 ) = - IE
          END IF
  160     CONTINUE
C
          DO 170 IE = 1 , NE
C
          IV1 = JE( 1 , IE )
          IV2 = JE( 2 , IE )
C
          IF( JV( 2 , IV1 ) . EQ . 0 ) THEN
          JV( 2 , IV1 ) = IE
          END IF
C
          IF( JV( 2 , IV2 ) . EQ . 0 ) THEN
          JV( 2 , IV2 ) = IE
          END IF
C
  170     CONTINUE
C
          DO 190 IS = 1 , NS
          SAREA( IS ) = 1. / XS( 3 , IS )
  190     CONTINUE
C
C=====================================================================
C
C --- OPTION FOR GLOBAL RECONNECTION ---------------------------------
C
          IF(IOPRCN.EQ.1)THEN
          DO 200 IE = 1 , NE
          CALL RECNC( IE , IDONE , ITL , ITR , JA , JB , JC , JD )
          CALL RECNC( JA , JADONE , ITL , ITR , JAA , JAB , JAC , JAD )
          CALL RECNC( JB , JBDONE , ITL , ITR , JBA , JBB , JBC , JBD )
          CALL RECNC( JC , JCDONE , ITL , ITR , JCA , JCB , JCC , JCD )
          CALL RECNC( JD , JDDONE , ITL , ITR , JDA , JDB , JDC , JDD )
  200     CONTINUE
          ENDIF
```

```
C
C=================================================================
C
C --- EXIT POINT FROM SUBROUTINE ----------------------------------
C
C     ------
      RETURN
C     ------
C
C     ---
      END
```

Thu Jul 1 14:15:40 1993 mainhd.f SUBROUTINE UPDATE

```
      SUBROUTINE UPDATE
C
C-----------------------------------------------------------I
C                                                           I
C       UPDATE BUFFERS IN A BLOCK OF NODES,EDGES, AND CELLS TO     I
C       CONSTRUCT THE NEW GEOMETRY FOR THE MESH.            I
C       THE BLOCKING SIZE IS DETERMINED BY PARAMETER ...MBL...     I
C                                                           I
C-----------------------------------------------------------I
C
C=================================================================
C
      include     'cmsh00.h'
      include     'chyd00.h'
      include     'cint00.h'
      include     'cphs10.h'
      include     'cphs20.h'
C
C=================================================================
C
C --- BREAK UP THE VERTEX, EDGE, AND TRIANGLE DATA INTO BLOCKS ---------
C
      NVECE = NE / MBL
      NREME = NE - NVECE * MBL
      NVECS = NS / MBL
      NREMS = NS - NVECS * MBL
      NVECV = NV / MBL
      NREMV = NV - NVECV * MBL
        PRINT *,NV,NE,NS,NVECE,NREME,NVECV,NREMV,NVECS,NREMS
C
      DO 105 INE = 1 , NVECE
      NOFVEE( INE ) = MBL
  105 CONTINUE
      NVEEE = NVECE
      IF( NREME . GT . 0 ) THEN
      NVEEE = NVECE + 1
      NOFVEE( NVEEE ) = NREME
      END IF
C
      DO 115 INS = 1 , NVECS
      NOFVES( INS ) = MBL
  115 CONTINUE
      NVEES = NVECS
      IF( NREMS . GT . 0 ) THEN
      NVEES = NVECS + 1
      NOFVES( NVEES ) = NREMS
      END IF
C
      DO 125 INV = 1 , NVECV
      NOFVEV( INV ) = MBL
  125 CONTINUE
      NVEEV = NVECV
      IF( NREMV . GT . 0 ) THEN
      NVEEV = NVECV + 1
      NOFVEV( NVEEV ) = NREMV
      END IF
C
C --- CALL TO THE GEOMETRY DEFINITION SUBROUTINE ----------------------
C
      CALL GEOMTR
```

```
C
C-----------------------------------------------------------------------
C
C --- EXIT POINT FROM SUBROUTINE ---------------------------------------
C
C     ------
      RETURN
C     ------
C
C     ---
      END
```

Thu Jul 1 14:15:40 1993 mainhd.f SUBROUTINE UPGRAD

```
      SUBROUTINE UPGRAD
C
C-----------------------------------------------------------------I
C                                                                  I
C      UPGRAD READS THE RESTART FILE FROM A PREVIOUS RUN           I
C      AND BREAKS THE DATA INTO BLOCKS AS DEFINED BY THE           I
C      PARAMETER ...MBL...                                         I
C                                                                  I
C-----------------------------------------------------------------I
C
      include     'cmsh00.h'
      include     'chyd00.h'
      include     'cint00.h'
      include     'cphs10.h'
      include     'cphs20.h'
C
C-----------------------------------------------------------------------
C
C      MVM MAX NUMBER OF VERTICES  (POINTS)
C      MEM MAX NUMBER OF EDGES     (INTERFACES)
C      MSM MAX NUMBER OF SIDES     (TRIANGLES)
C
C-----------------------------------------------------------------------
C
C
       READ (88) NV,NVMK,NE,NEMK,NS,NSMK,ITERAT
C
C --- READ IN VERTEX INFORMATION ----------------------------------------
C
        READ (88) ((JV(IK,IV),IK=1,2),(XV(IK,IV),IK=1,2),IV=1,NV)
C
C --- READ IN EDGE INFORMATION ( EDGES OF TRIANGLES).-------------------
C
        READ (88) ((JE(KK,IE),KK=1,5),(XE(KI,IE),KI=1,2),IE=1,NE)
        READ (88) (XN(IE),YN(IE),XXN(IE),YYN(IE),IE=1,NE)
C
C --- READ IN SIDE (TRIANGLE) INFORMATION.------------------------------
C
         READ (88) ((JS(KK,IS),KK=1,6),(XS(KI,IS),KI=1,3),IS=1,NS)
         READ (88) (XMIDL(IE),YMIDL(IE),XYMIDL(IE),IE=1,NE)
         READ (88) SAREVG,NVECE,NREME,NVECV,NREMV,NVECS,NREMS
C
C --- PRINT PROMT TO CONSOLE -------------------------------------------
C
         PRINT * , NE,NS
C
C --- DEFINE INVERSE AREA OF TRIANGLES ---------------------------------
C
      DO 100 IS = 1 , NS
      SAREA( IS ) = 1. / XS( 3 , IS )
  100 CONTINUE
C
C --- BREAKUP THE DATA STRUCTURE INTO BLOCKS ---------------------------
C
      DO 105 INE = 1 , NVECE
      NOFVEE( INE ) = MBL
  105 CONTINUE
      NVEEE = NVECE
      IF( NREME . GT . 0 ) THEN
      NVEEE = NVECE + 1
```

```
            NOFVEE( NVEEE ) = NREME
            END IF
      C
            DO 115 INS = 1 , NVECS
            NOFVES( INS ) = MBL
        115 CONTINUE
            NVEES = NVECS
            IF( NREMS . GT . 0 ) THEN
            NVEES = NVECS + 1
            NOFVES( NVEES ) = NREMS
            END IF
      C
            DO 125 INV = 1 , NVECV
            NOFVEV( INV ) = MBL
        125 CONTINUE
            NVEEV = NVECV
            IF( NREMV . GT . 0 ) THEN
            NVEEV = NVECV + 1
            NOFVEV( NVEEV ) = NREMV
            END IF
      C
      C --- PRINTOUT THE VERTEX,EDGE, AND TRIANGLE BLOCK DATA ----------------
      C
              PRINT *,NV,NE,NS,NVECE,NREME,NVECV,NREMV,NVECS,NREMS
      C
      C======================================================================
      C
      C --- EXIT POINT FROM SUBROUTINE --------------------------------------
      C
      C     ------
            RETURN
      C     ------
      C
      C     ---
            END
```

Thu Jul 1 14:15:55 1993 gradhd.f Module List - alphabetical order

```
      SUBROUTINE GRDFLX
C
C-------------------------------------------------------------I
C                                                             I
C       GRAFLX COMPUTE THE GRADIENT FOR ERROR INDICATOR       I
C               CALCULATION FOR STEADY STATE                  I
C                                                             I
C-------------------------------------------------------------I
C
      include     'cmsh00.h'
      include     'chyd00.h'
      include     'cint00.h'
      include     'cphs10.h'
      include     'cphs20.h'
C
C=====================================================================
C
      REAL AA(3,3),BB(3,4),B(3),INDX(3),ATEMP(3,3,3),BTEMP(3,4,3)
      REAL AA0(3,3),BB0(3,4)
C
C=====================================================================
C
C
C --- BEGIN LOOP OVER ALL CELLS IN THE DOMAIN -------------------------
C
      NS1 = 1
      NS2 = NOFVES( 1 )
      DO 90 INS = 1 , NVEES
C
C --- FETCH HYDRO QUANTITIES ------------------------------------------
C
      DO 105 IS = NS1 , NS2
          KS = IS - NS1 + 1
C
       XSM = XS( 1 , IS )
       YSM = XS( 2 , IS )
       XSM2 = XSM * XSM
       YSM2 = YSM * YSM
       XYSM = XSM * YSM
C
       AA0( 1 , 1 ) = 1.0
       AA0( 1 , 2 ) = XSM
       AA0( 1 , 3 ) = YSM
C
       AA0( 2 , 1 ) = XSM
       AA0( 2 , 2 ) = XSM2
       AA0( 2 , 3 ) = XYSM
C
       AA0( 3 , 1 ) = YSM
       AA0( 3 , 2 ) = XYSM
       AA0( 3 , 3 ) = YSM2
C
       BB1 = HYDV( IS , 4 )
       BB2 = SQRT ( ( HYDV( IS , 2 ) *  HYDV( IS , 2 ) +
     .                HYDV( IS , 3 ) *  HYDV( IS , 3 ) ) *
     .         HYDV( IS , 1 ) /  HYDV( IS , 4 ) / HYDV( IS , 5 ) )
C
       BB1X = BB1 * XSM
       BB2X = BB2 * XSM
C
       BB1Y = BB1 * YSM
       BB2Y = BB2 * YSM
C
       BB0( 1 , 1 ) = BB1
       BB0( 1 , 2 ) = BB2
C
       BB0( 2 , 1 ) = BB1X
       BB0( 2 , 2 ) = BB2X
C
       BB0( 3 , 1 ) = BB1Y
       BB0( 3 , 2 ) = BB2Y
C
       DO 115 IK = 1 , 3
```

```
      IE = JS( IK + 3 , IS )
      IF( IE . GT . 0 ) THEN
      ISS = JE( 4 , IE )
      ELSE
      ISS = JE( 3 , -IE )
      END IF
C
      IF( ISS . NE . 0 ) THEN
      XSS = XS( 1 , ISS )
      YSS = XS( 2 , ISS )
C
      HYDVP = HYDV( ISS , 4 )
      HYDVR = SQRT ( ( HYDV( ISS , 2 ) *  HYDV( ISS , 2 ) +
     .                 HYDV( ISS , 3 ) *  HYDV( ISS , 3 ) ) *
     .         HYDV( ISS , 1 ) /  HYDV( ISS , 4 ) / HYDV( ISS , 5 ) )
C
      ELSE
C
      IE = IABS( IE )
      XSS = 2. * XMIDL( IE ) - XSM
      YSS = 2. * YMIDL( IE ) - YSM
C
      HYDVP = BB1
      HYDVR = BB2
      IJE5 = JE( 5 , IE )
      IF( IJE5 . EQ . 8 ) THEN
      HYDVP =   PIN
      HYDVR =   SQRT( UVIN * UVIN * RIN / PIN / HRGG )
      END IF
C
      END IF
C
      XSS2 = XSS * XSS
      YSS2 = YSS * YSS
      XYSS = XSS * YSS
C
      ATEMP( 1 , 1 , IK ) = 1.0
      ATEMP( 1 , 2 , IK ) = XSS
      ATEMP( 1 , 3 , IK ) = YSS
C
      ATEMP( 2 , 1 , IK ) = XSS
      ATEMP( 2 , 2 , IK ) = XSS2
      ATEMP( 2 , 3 , IK ) = XYSS
C
      ATEMP( 3 , 1 , IK ) = YSS
      ATEMP( 3 , 2 , IK ) = XYSS
      ATEMP( 3 , 3 , IK ) = YSS2
C
      BTEMP( 1 , 1 , IK ) = HYDVP
      BTEMP( 1 , 2 , IK ) = HYDVR
C
      BTEMP( 2 , 1 , IK ) = HYDVP * XSS
      BTEMP( 2 , 2 , IK ) = HYDVR * XSS
C
      BTEMP( 3 , 1 , IK ) = HYDVP * YSS
      BTEMP( 3 , 2 , IK ) = HYDVR * YSS
C
  115 CONTINUE
C
      AA( 1 , 1 ) =   AA0( 1 , 1 )     + ATEMP( 1 , 1 , 1 ) +
     .            ATEMP( 1 , 1 , 2 ) + ATEMP( 1 , 1 , 3 )
      AA( 1 , 2 ) =   AA0( 1 , 2 )     + ATEMP( 1 , 2 , 1 ) +
     .            ATEMP( 1 , 2 , 2 ) + ATEMP( 1 , 2 , 3 )
      AA( 1 , 3 ) =   AA0( 1 , 3 )     + ATEMP( 1 , 3 , 1 ) +
     .            ATEMP( 1 , 3 , 2 ) + ATEMP( 1 , 3 , 3 )
C
      AA( 2 , 1 ) =   AA0( 2 , 1 )     + ATEMP( 2 , 1 , 1 ) +
     .            ATEMP( 2 , 1 , 2 ) + ATEMP( 2 , 1 , 3 )
      AA( 2 , 2 ) =   AA0( 2 , 2 )     + ATEMP( 2 , 2 , 1 ) +
     .            ATEMP( 2 , 2 , 2 ) + ATEMP( 2 , 2 , 3 )
      AA( 2 , 3 ) =   AA0( 2 , 3 )     + ATEMP( 2 , 3 , 1 ) +
     .            ATEMP( 2 , 3 , 2 ) + ATEMP( 2 , 3 , 3 )
C
      AA( 3 , 1 ) =   AA0( 3 , 1 )     + ATEMP( 3 , 1 , 1 ) +
```

```
     .                    ATEMP( 3 , 1 , 2 ) + ATEMP( 3 , 1 , 3 )
        AA( 3 , 2 ) =   AA0( 3 , 2 )      + ATEMP( 3 , 2 , 1 ) +
     .                    ATEMP( 3 , 2 , 2 ) + ATEMP( 3 , 2 , 3 )
        AA( 3 , 3 ) =   AA0( 3 , 3 )      + ATEMP( 3 , 3 , 1 ) +
     .                    ATEMP( 3 , 3 , 2 ) + ATEMP( 3 , 3 , 3 )
C
        BB( 1 , 1 ) =   BB0( 1 , 1 )      + BTEMP( 1 , 1 , 1 ) +
     .                    BTEMP( 1 , 1 , 2 ) + BTEMP( 1 , 1 , 3 )
        BB( 1 , 2 ) =   BB0( 1 , 2 )      + BTEMP( 1 , 2 , 1 ) +
     .                    BTEMP( 1 , 2 , 2 ) + BTEMP( 1 , 2 , 3 )
C
        BB( 2 , 1 ) =   BB0( 2 , 1 )      + BTEMP( 2 , 1 , 1 ) +
     .                    BTEMP( 2 , 1 , 2 ) + BTEMP( 2 , 1 , 3 )
        BB( 2 , 2 ) =   BB0( 2 , 2 )      + BTEMP( 2 , 2 , 1 ) +
     .                    BTEMP( 2 , 2 , 2 ) + BTEMP( 2 , 2 , 3 )
C
        BB( 3 , 1 ) =   BB0( 3 , 1 )      + BTEMP( 3 , 1 , 1 ) +
     .                    BTEMP( 3 , 1 , 2 ) + BTEMP( 3 , 1 , 3 )
        BB( 3 , 2 ) =   BB0( 3 , 2 )      + BTEMP( 3 , 2 , 1 ) +
     .                    BTEMP( 3 , 2 , 2 ) + BTEMP( 3 , 2 , 3 )
C
        DETERM = AA( 1 , 1 ) * ( AA( 2 , 2 ) * AA( 3 , 3 ) -
     .                           AA( 3 , 2 ) * AA( 2 , 3 ) ) +
     .           AA( 2 , 1 ) * ( AA( 3 , 2 ) * AA( 1 , 3 ) -
     .                           AA( 1 , 2 ) * AA( 3 , 3 ) ) +
     .           AA( 3 , 1 ) * ( AA( 1 , 2 ) * AA( 2 , 3 ) -
     .                           AA( 2 , 2 ) * AA( 1 , 3 ) )
C
        DTRMIN = 1. / DETERM
C
        AAA1 = AA( 2 , 3 ) * AA( 3 , 1 ) - AA( 2 , 1 ) * AA( 3 , 3 )
        AAA2 = AA( 3 , 3 ) * AA( 1 , 1 ) - AA( 3 , 1 ) * AA( 1 , 3 )
        AAA3 = AA( 1 , 3 ) * AA( 2 , 1 ) - AA( 1 , 1 ) * AA( 2 , 3 )
C
        AAA4 = AA( 2 , 1 ) * AA( 3 , 2 ) - AA( 3 , 1 ) * AA( 2 , 2 )
        AAA5 = AA( 3 , 1 ) * AA( 1 , 2 ) - AA( 1 , 1 ) * AA( 3 , 2 )
        AAA6 = AA( 1 , 1 ) * AA( 2 , 2 ) - AA( 2 , 1 ) * AA( 1 , 2 )
C
        PL( IS ) = DTRMIN * ( BB( 1 , 1 ) * AAA1 +
     .                        BB( 2 , 1 ) * AAA2 +
     .                        BB( 3 , 1 ) * AAA3 )
C
        PR( IS ) = DTRMIN * ( BB( 1 , 1 ) * AAA4 +
     .                        BB( 2 , 1 ) * AAA5 +
     .                        BB( 3 , 1 ) * AAA6 )
C
        RL( IS ) = DTRMIN * ( BB( 1 , 2 ) * AAA1 +
     .                        BB( 2 , 2 ) * AAA2 +
     .                        BB( 3 , 2 ) * AAA3 )
C
        RR( IS ) = DTRMIN * ( BB( 1 , 2 ) * AAA4 +
     .                        BB( 2 , 2 ) * AAA5 +
     .                        BB( 3 , 2 ) * AAA6 )
C
 105  CONTINUE
C
      NS1 = NS2 + 1
      NS2 = NS2 + NOFVES( INS + 1 )
 90   CONTINUE
C
C===================================================================
C
C --- EXIT POINT FROM SUBROUTINE ------------------------------------
C
C     ------
      RETURN
C     ------
C
C     ---
      END
```

```
      SUBROUTINE GRDENG
C
C-------------------------------------------------------------------I
C                                                                   I
C     GRAENG COMPUTE THE GRADIENT FOR SECOND ORDER CALCULATION      I
C                                                                   I
C-------------------------------------------------------------------I
C
      include    'cmsh00.h'
      include    'chyd00.h'
      include    'cint00.h'
      include    'cphs10.h'
      include    'cphs20.h'
C
C=======================================================================
C
      REAL RRMIDL(MBP),PPMIDL(MBP),UUMIDL(MBP),VVMIDL(MBP)
      REAL RIGRAD(MBP),PIGRAD(MBP),UIGRAD(MBP),VIGRAD(MBP)
      REAL RJGRAD(MBP),PJGRAD(MBP),UJGRAD(MBP),VJGRAD(MBP)
      REAL AA(3,3),BB(3,4),B(3),INDX(3),ATEMP(3,3,3),BTEMP(3,4,3)
      REAL AA0(3,3),BB0(3,4)
C
C=======================================================================
C
C --- BEGIN LOOP OVER ALL CELLS IN THE DOMAIN --------------------------
C
      NS1 = 1
      NS2 = NOFVES( 1 )
      DO 90 INS = 1 , NVEES
C
C --- FETCH HYDRO QUANTITIES -------------------------------------------
C
      DO 105 IS = NS1 , NS2
C
       XSM = XS( 1 , IS )
       YSM = XS( 2 , IS )
       XSM2 = XSM * XSM
       YSM2 = YSM * YSM
       XYSM = XSM * YSM
C
       AA0( 1 , 1 ) = 1.0
       AA0( 1 , 2 ) = XSM
       AA0( 1 , 3 ) = YSM
C
       AA0( 2 , 1 ) = XSM
       AA0( 2 , 2 ) = XSM2
       AA0( 2 , 3 ) = XYSM
C
       AA0( 3 , 1 ) = YSM
       AA0( 3 , 2 ) = XYSM
       AA0( 3 , 3 ) = YSM2
C
       BB1 = HYDFLX( IS , 1 )
       BB2 = HYDFLX( IS , 2 )
       BB3 = HYDFLX( IS , 4 )
C
       BB1X = BB1 * XSM
       BB2X = BB2 * XSM
       BB3X = BB3 * XSM
C
       BB1Y = BB1 * YSM
       BB2Y = BB2 * YSM
       BB3Y = BB3 * YSM
C
       BB0( 1 , 1 ) = BB1
       BB0( 1 , 2 ) = BB2
       BB0( 1 , 3 ) = BB3
C
       BB0( 2 , 1 ) = BB1X
       BB0( 2 , 2 ) = BB2X
       BB0( 2 , 3 ) = BB3X
C
       BB0( 3 , 1 ) = BB1Y
       BB0( 3 , 2 ) = BB2Y
```

```
      BB0( 3 , 3 ) = BB3Y
C
      DO 115 IK = 1 , 3
      IE = JS( IK + 3 , IS )
      IF( IE . GT . 0 ) THEN
      ISS = JE( 4 , IE )
      ELSE
      ISS = JE( 3 , - IE )
      END IF
C
      IF( ISS . NE . 0 ) THEN
      XSS = XS( 1 , ISS )
      YSS = XS( 2 , ISS )
C
      HYDVR = HYDFLX( ISS , 1 )
      HYDVU = HYDFLX( ISS , 2 )
      HYDVV = HYDFLX( ISS , 4 )
C
      ELSE
C
      IE = IABS( IE )
      HYDVR = BB1
      HYDVU = BB2
      HYDVV = BB3
C
      XSS = 2. * XMIDL( IE ) - XSM
      YSS = 2. * YMIDL( IE ) - YSM
C
      END IF
C
      XSS2 = XSS * XSS
      YSS2 = YSS * YSS
      XYSS = XSS * YSS
C
      ATEMP( 1 , 1 , IK ) = 1.0
      ATEMP( 1 , 2 , IK ) = XSS
      ATEMP( 1 , 3 , IK ) = YSS
C
      ATEMP( 2 , 1 , IK ) = XSS
      ATEMP( 2 , 2 , IK ) = XSS2
      ATEMP( 2 , 3 , IK ) = XYSS
C
      ATEMP( 3 , 1 , IK ) = YSS
      ATEMP( 3 , 2 , IK ) = XYSS
      ATEMP( 3 , 3 , IK ) = YSS2
C
      BTEMP( 1 , 1 , IK ) = HYDVR
      BTEMP( 1 , 2 , IK ) = HYDVU
      BTEMP( 1 , 3 , IK ) = HYDVV
C
      BTEMP( 2 , 1 , IK ) = HYDVR * XSS
      BTEMP( 2 , 2 , IK ) = HYDVU * XSS
      BTEMP( 2 , 3 , IK ) = HYDVV * XSS
C
      BTEMP( 3 , 1 , IK ) = HYDVR * YSS
      BTEMP( 3 , 2 , IK ) = HYDVU * YSS
      BTEMP( 3 , 3 , IK ) = HYDVV * YSS
C
  115 CONTINUE
C
      AA( 1 , 1 ) =  AA0( 1 , 1 )     + ATEMP( 1 , 1 , 1 ) +
     .               ATEMP( 1 , 1 , 2 ) + ATEMP( 1 , 1 , 3 )
      AA( 1 , 2 ) =  AA0( 1 , 2 )     + ATEMP( 1 , 2 , 1 ) +
     .               ATEMP( 1 , 2 , 2 ) + ATEMP( 1 , 2 , 3 )
      AA( 1 , 3 ) =  AA0( 1 , 3 )     + ATEMP( 1 , 3 , 1 ) +
     .               ATEMP( 1 , 3 , 2 ) + ATEMP( 1 , 3 , 3 )
C
      AA( 2 , 1 ) =  AA0( 2 , 1 )     + ATEMP( 2 , 1 , 1 ) +
     .               ATEMP( 2 , 1 , 2 ) + ATEMP( 2 , 1 , 3 )
      AA( 2 , 2 ) =  AA0( 2 , 2 )     + ATEMP( 2 , 2 , 1 ) +
     .               ATEMP( 2 , 2 , 2 ) + ATEMP( 2 , 2 , 3 )
      AA( 2 , 3 ) =  AA0( 2 , 3 )     + ATEMP( 2 , 3 , 1 ) +
     .               ATEMP( 2 , 3 , 2 ) + ATEMP( 2 , 3 , 3 )
C
```

```fortran
      AA( 3 , 1 ) =  AA0( 3 , 1 )     + ATEMP( 3 , 1 , 1 ) +
     .              ATEMP( 3 , 1 , 2 ) + ATEMP( 3 , 1 , 3 )
      AA( 3 , 2 ) =  AA0( 3 , 2 )     + ATEMP( 3 , 2 , 1 ) +
     .              ATEMP( 3 , 2 , 2 ) + ATEMP( 3 , 2 , 3 )
      AA( 3 , 3 ) =  AA0( 3 , 3 )     + ATEMP( 3 , 3 , 1 ) +
     .              ATEMP( 3 , 3 , 2 ) + ATEMP( 3 , 3 , 3 )
C
      BB( 1 , 1 ) =  BB0( 1 , 1 )     + BTEMP( 1 , 1 , 1 ) +
     .              BTEMP( 1 , 1 , 2 ) + BTEMP( 1 , 1 , 3 )
      BB( 1 , 2 ) =  BB0( 1 , 2 )     + BTEMP( 1 , 2 , 1 ) +
     .              BTEMP( 1 , 2 , 2 ) + BTEMP( 1 , 2 , 3 )
      BB( 1 , 3 ) =  BB0( 1 , 3 )     + BTEMP( 1 , 3 , 1 ) +
     .              BTEMP( 1 , 3 , 2 ) + BTEMP( 1 , 3 , 3 )
C
      BB( 2 , 1 ) =  BB0( 2 , 1 )     + BTEMP( 2 , 1 , 1 ) +
     .              BTEMP( 2 , 1 , 2 ) + BTEMP( 2 , 1 , 3 )
      BB( 2 , 2 ) =  BB0( 2 , 2 )     + BTEMP( 2 , 2 , 1 ) +
     .              BTEMP( 2 , 2 , 2 ) + BTEMP( 2 , 2 , 3 )
      BB( 2 , 3 ) =  BB0( 2 , 3 )     + BTEMP( 2 , 3 , 1 ) +
     .              BTEMP( 2 , 3 , 2 ) + BTEMP( 2 , 3 , 3 )
C
      BB( 3 , 1 ) =  BB0( 3 , 1 )     + BTEMP( 3 , 1 , 1 ) +
     .              BTEMP( 3 , 1 , 2 ) + BTEMP( 3 , 1 , 3 )
      BB( 3 , 2 ) =  BB0( 3 , 2 )     + BTEMP( 3 , 2 , 1 ) +
     .              BTEMP( 3 , 2 , 2 ) + BTEMP( 3 , 2 , 3 )
      BB( 3 , 3 ) =  BB0( 3 , 3 )     + BTEMP( 3 , 3 , 1 ) +
     .              BTEMP( 3 , 3 , 2 ) + BTEMP( 3 , 3 , 3 )
C
      DETERM = AA( 1 , 1 ) * ( AA( 2 , 2 ) * AA( 3 , 3 ) -
     .                         AA( 3 , 2 ) * AA( 2 , 3 ) ) +
     .         AA( 2 , 1 ) * ( AA( 1 , 3 ) * AA( 3 , 2 ) -
     .                         AA( 3 , 3 ) * AA( 1 , 2 ) ) +
     .         AA( 3 , 1 ) * ( AA( 1 , 2 ) * AA( 2 , 3 ) -
     .                         AA( 2 , 2 ) * AA( 1 , 3 ) )
C
      DTRMIN = 1. / DETERM
C
      AAA1 = AA( 2 , 3 ) * AA( 3 , 1 ) - AA( 2 , 1 ) * AA( 3 , 3 )
      AAA2 = AA( 3 , 3 ) * AA( 1 , 1 ) - AA( 3 , 1 ) * AA( 1 , 3 )
      AAA3 = AA( 1 , 3 ) * AA( 2 , 1 ) - AA( 1 , 1 ) * AA( 2 , 3 )
C
      AAA4 = AA( 2 , 1 ) * AA( 3 , 2 ) - AA( 3 , 1 ) * AA( 2 , 2 )
      AAA5 = AA( 3 , 1 ) * AA( 1 , 2 ) - AA( 1 , 1 ) * AA( 3 , 2 )
      AAA6 = AA( 1 , 1 ) * AA( 2 , 2 ) - AA( 2 , 1 ) * AA( 1 , 2 )
C
      RR( IS ) = DTRMIN * ( BB( 1 , 1 ) * AAA1 +
     .                      BB( 2 , 1 ) * AAA2 +
     .                      BB( 3 , 1 ) * AAA3 )
C
      RL( IS ) = DTRMIN * ( BB( 1 , 1 ) * AAA4 +
     .                      BB( 2 , 1 ) * AAA5 +
     .                      BB( 3 , 1 ) * AAA6 )
C
      UR( IS ) = DTRMIN * ( BB( 1 , 2 ) * AAA1 +
     .                      BB( 2 , 2 ) * AAA2 +
     .                      BB( 3 , 2 ) * AAA3 )
C
      UL( IS ) = DTRMIN * ( BB( 1 , 2 ) * AAA4 +
     .                      BB( 2 , 2 ) * AAA5 +
     .                      BB( 3 , 2 ) * AAA6 )
C
      VR( IS ) = DTRMIN * ( BB( 1 , 3 ) * AAA1 +
     .                      BB( 2 , 3 ) * AAA2 +
     .                      BB( 3 , 3 ) * AAA3 )
C
      VL( IS ) = DTRMIN * ( BB( 1 , 3 ) * AAA4 +
     .                      BB( 2 , 3 ) * AAA5 +
     .                      BB( 3 , 3 ) * AAA6 )
C
  105 CONTINUE
C
      NS1 = NS2 + 1
      NS2 = NS2 + NOFVES( INS + 1 )
   90 CONTINUE
```

```
C
C================================================================
C
C --- EXIT POINT FROM SUBROUTINE --------------------------------------
C
C      ------
       RETURN
C      ------
C
C      ---
       END
```

Thu Jul 1 14:15:55 1993 gradhd.f SUBROUTINE GRADNL

```
       SUBROUTINE GRADNL
C
C-------------------------------------------------------------I
C                                                             I
C       GRADNL COMPUTE THE GRADIENT FOR SECOND ORDER CALCULATION   I
C              SEARCH FOR ALL TRIANGLES SUROUNDING THE TARGET      I
C              CELL FOR COMPUTING THE GRADIENT APPLYING LEAST      I
C              SQUARE TECHNIQUE                                    I
C                                                             I
C-------------------------------------------------------------I
C
       include     'cmsh00.h'
       include     'chyd00.h'
       include     'cint00.h'
       include     'cphs10.h'
       include     'cphs20.h'
C
C================================================================
C
       REAL RRMIDL(MBP),PPMIDL(MBP),UUMIDL(MBP),VVMIDL(MBP)
       REAL RIGRAD(MBP),PIGRAD(MBP),UIGRAD(MBP),VIGRAD(MBP)
       REAL RJGRAD(MBP),PJGRAD(MBP),UJGRAD(MBP),VJGRAD(MBP)
       REAL RMAX(MBP),PMAX(MBP),UMAX(MBP),VMAX(MBP)
       REAL RMIN(MBP),PMIN(MBP),UMIN(MBP),VMIN(MBP)
       REAL RLEFTT(MBP),ULEFTT(MBP),VLEFTT(MBP),PLEFTT(MBP)
       REAL RRIGHT(MBP),URIGHT(MBP),VRIGHT(MBP),PRIGHT(MBP)
       REAL ROR(3),UOR(3),VOR(3),POR(3)
       REAL ROL(3),UOL(3),VOL(3),POL(3)
       REAL AA(3,3),BB(3,4),B(3),INDX(3),ATEMP(3,3),BTEMP(3,4)
C
C================================================================
C
C --- BEGIN LOOP OVER ALL CELLS IN THE DOMAIN ------------------------
C
       NS1 = 1
       NS2 = NOFVES( 1 )
       DO 90 INS = 1 , NVEES
C
C --- FETCH HYDRO QUANTITIES -------------------------------------------
C
       DO 105 IS = NS1 , NS2
C
        JJCOLR = 0
C
        DO 115 IK = 1 , 3
        IVV = JS( IK , IS )
        IEE = JV( 2 , IVV )
C
        IF( IEE . GT . 0 ) THEN
C
        IV1 = JE( 1 , IEE )
        IF( IV1 . EQ . IVV ) THEN
        ISI = JE( 3 , IEE )
        ELSE
        ISI = JE( 4 , IEE )
        END IF
        ISS = ISI
        IE = IEE
C
  150   CONTINUE
```

```
C
      JJCOLR = JJCOLR + 1
      IICOLR( JJCOLR ) = ISS
C
      DO 160 IR = 1 , 3
      JR = MOD( IR , 3 ) + 1
      IEA = IABS( JS( JR + 3 , ISS ) )
      IF( IEA . EQ . IE ) THEN
      JJR = MOD( JR + 1 , 3 ) + 4
      IER = IABS( JS( JJR , ISS ) )
C
      IV1 = JE( 1 , IER )
      IF( IV1 . EQ . IVV ) THEN
      ISR = JE( 3 , IER )
      ELSE
      ISR = JE( 4 , IER )
      END IF
      END IF
C
  160 CONTINUE
C
      IF( ISR . NE . ISI ) THEN
      ISS = ISR
      IE = IER
      GO TO 150
      END IF
C
      ELSE
C
      IEE = - IEE
      IV1 = JE( 1 , IEE )
      IF( IV1 . EQ . IVV ) THEN
      ISI = JE( 3 , IEE )
      ELSE
      ISI = JE( 4 , IEE )
      END IF
      ISS = ISI
      ISI = 0
      IE = IEE
C
  170 CONTINUE
C
      JJCOLR = JJCOLR + 1
      IICOLR( JJCOLR ) = ISS
C
      DO 180 IR = 1 , 3
      JR = MOD( IR , 3 ) + 1
      IEA = IABS( JS( JR + 3 , ISS ) )
      IF( IEA . EQ . IE ) THEN
      JJR = MOD( JR + 1 , 3 ) + 4
      IER = IABS( JS( JJR , ISS ) )
C
      IV1 = JE( 1 , IER )
      IF( IV1 . EQ . IVV ) THEN
      ISR = JE( 3 , IER )
      ELSE
      ISR = JE( 4 , IER )
      END IF
      END IF
C
  180 CONTINUE
C
      IF( ISR . NE . ISI ) THEN
      ISS = ISR
      IE = IER
      GO TO 170
      END IF
C
      END IF
  115 CONTINUE
C
      ATEMP( 1 , 1 ) = 0.
      ATEMP( 1 , 2 ) = 0.
      ATEMP( 1 , 3 ) = 0.
```

```
C
        ATEMP( 2 , 1 ) = 0.
        ATEMP( 2 , 2 ) = 0.
        ATEMP( 2 , 3 ) = 0.
C
        ATEMP( 3 , 1 ) = 0.
        ATEMP( 3 , 2 ) = 0.
        ATEMP( 3 , 3 ) = 0.
C
        BTEMP( 1 , 1 ) = 0.
        BTEMP( 1 , 2 ) = 0.
        BTEMP( 1 , 3 ) = 0.
        BTEMP( 1 , 4 ) = 0.
C
        BTEMP( 2 , 1 ) = 0.
        BTEMP( 2 , 2 ) = 0.
        BTEMP( 2 , 3 ) = 0.
        BTEMP( 2 , 4 ) = 0.
C
        BTEMP( 3 , 1 ) = 0.
        BTEMP( 3 , 2 ) = 0.
        BTEMP( 3 , 3 ) = 0.
        BTEMP( 3 , 4 ) = 0.
C
        DO 225 KK = 2 , JJCOLR
        ISS = IICOLR( KK )
        IF( ISS . NE . 0 ) THEN
        XSS = XS( 1 , ISS )
        YSS = XS( 2 , ISS )
C
        HYDVR = HYDV( ISS , 1 )
        HYDVU = HYDV( ISS , 2 )
        HYDVV = HYDV( ISS , 3 )
        HYDVP = HYDV( ISS , 4 )
C
        XSS2 = XSS * XSS
        YSS2 = YSS * YSS
        XYSS = XSS * YSS
C
        ATEMP( 1 , 1 ) = ATEMP( 1 , 1 ) + 1.0
        ATEMP( 1 , 2 ) = ATEMP( 1 , 2 ) + XSS
        ATEMP( 1 , 3 ) = ATEMP( 1 , 3 ) + YSS
C
        ATEMP( 2 , 1 ) = ATEMP( 2 , 1 ) + XSS
        ATEMP( 2 , 2 ) = ATEMP( 2 , 2 ) + XSS2
        ATEMP( 2 , 3 ) = ATEMP( 2 , 3 ) + XYSS
C
        ATEMP( 3 , 1 ) = ATEMP( 3 , 1 ) + YSS
        ATEMP( 3 , 2 ) = ATEMP( 3 , 2 ) + XYSS
        ATEMP( 3 , 3 ) = ATEMP( 3 , 3 ) + YSS2
C
        BTEMP( 1 , 1 ) = BTEMP( 1 , 1 ) + HYDVR
        BTEMP( 1 , 2 ) = BTEMP( 1 , 2 ) + HYDVU
        BTEMP( 1 , 3 ) = BTEMP( 1 , 3 ) + HYDVV
        BTEMP( 1 , 4 ) = BTEMP( 1 , 4 ) + HYDVP
C
        BTEMP( 2 , 1 ) = BTEMP( 2 , 1 ) + HYDVR * XSS
        BTEMP( 2 , 2 ) = BTEMP( 2 , 2 ) + HYDVU * XSS
        BTEMP( 2 , 3 ) = BTEMP( 2 , 3 ) + HYDVV * XSS
        BTEMP( 2 , 4 ) = BTEMP( 2 , 4 ) + HYDVP * XSS
C
        BTEMP( 3 , 1 ) = BTEMP( 3 , 1 ) + HYDVR * YSS
        BTEMP( 3 , 2 ) = BTEMP( 3 , 2 ) + HYDVU * YSS
        BTEMP( 3 , 3 ) = BTEMP( 3 , 3 ) + HYDVV * YSS
        BTEMP( 3 , 4 ) = BTEMP( 3 , 4 ) + HYDVP * YSS
C
        END IF
C
  225 CONTINUE
C
        AA( 1 , 1 ) =  ATEMP( 1 , 1 )
        AA( 1 , 2 ) =  ATEMP( 1 , 2 )
        AA( 1 , 3 ) =  ATEMP( 1 , 3 )
C
```

```
            AA( 2 , 1 ) =   ATEMP( 2 , 1 )
            AA( 2 , 2 ) =   ATEMP( 2 , 2 )
            AA( 2 , 3 ) =   ATEMP( 2 , 3 )
C
            AA( 3 , 1 ) =   ATEMP( 3 , 1 )
            AA( 3 , 2 ) =   ATEMP( 3 , 2 )
            AA( 3 , 3 ) =   ATEMP( 3 , 3 )
C
            BB( 1 , 1 ) =   BTEMP( 1 , 1 )
            BB( 1 , 2 ) =   BTEMP( 1 , 2 )
            BB( 1 , 3 ) =   BTEMP( 1 , 3 )
            BB( 1 , 4 ) =   BTEMP( 1 , 4 )
C
            BB( 2 , 1 ) =   BTEMP( 2 , 1 )
            BB( 2 , 2 ) =   BTEMP( 2 , 2 )
            BB( 2 , 3 ) =   BTEMP( 2 , 3 )
            BB( 2 , 4 ) =   BTEMP( 2 , 4 )
C
            BB( 3 , 1 ) =   BTEMP( 3 , 1 )
            BB( 3 , 2 ) =   BTEMP( 3 , 2 )
            BB( 3 , 3 ) =   BTEMP( 3 , 3 )
            BB( 3 , 4 ) =   BTEMP( 3 , 4 )
C
            DETERM = AA( 1 , 1 ) * ( AA( 2 , 2 ) * AA( 3 , 3 ) -
     .                               AA( 3 , 2 ) * AA( 2 , 3 ) ) +
     .               AA( 2 , 1 ) * ( AA( 1 , 3 ) * AA( 3 , 2 ) -
     .                               AA( 3 , 3 ) * AA( 1 , 2 ) ) +
     .               AA( 3 , 1 ) * ( AA( 1 , 2 ) * AA( 2 , 3 ) -
     .                               AA( 2 , 2 ) * AA( 1 , 3 ) )
C
            DTRMIN = 1. / DETERM
C
            AAA1 = AA( 2 , 3 ) * AA( 3 , 1 ) - AA( 2 , 1 ) * AA( 3 , 3 )
            AAA2 = AA( 3 , 3 ) * AA( 1 , 1 ) - AA( 3 , 1 ) * AA( 1 , 3 )
            AAA3 = AA( 1 , 3 ) * AA( 2 , 1 ) - AA( 1 , 1 ) * AA( 2 , 3 )
C
            AAA4 = AA( 2 , 1 ) * AA( 3 , 2 ) - AA( 3 , 1 ) * AA( 2 , 2 )
            AAA5 = AA( 3 , 1 ) * AA( 1 , 2 ) - AA( 1 , 1 ) * AA( 3 , 2 )
            AAA6 = AA( 1 , 1 ) * AA( 2 , 2 ) - AA( 2 , 1 ) * AA( 1 , 2 )
C
            RGRAD( IS , 1 ) = DTRMIN * ( BB( 1 , 1 ) * AAA1 +
     .                                   BB( 2 , 1 ) * AAA2 +
     .                                   BB( 3 , 1 ) * AAA3 )
C
            RGRAD( IS , 2 ) = DTRMIN * ( BB( 1 , 1 ) * AAA4 +
     .                                   BB( 2 , 1 ) * AAA5 +
     .                                   BB( 3 , 1 ) * AAA6 )
C
            UGRAD( IS , 1 ) = DTRMIN * ( BB( 1 , 2 ) * AAA1 +
     .                                   BB( 2 , 2 ) * AAA2 +
     .                                   BB( 3 , 2 ) * AAA3 )
C
            UGRAD( IS , 2 ) = DTRMIN * ( BB( 1 , 2 ) * AAA4 +
     .                                   BB( 2 , 2 ) * AAA5 +
     .                                   BB( 3 , 2 ) * AAA6 )
C
            VGRAD( IS , 1 ) = DTRMIN * ( BB( 1 , 3 ) * AAA1 +
     .                                   BB( 2 , 3 ) * AAA2 +
     .                                   BB( 3 , 3 ) * AAA3 )
C
            VGRAD( IS , 2 ) = DTRMIN * ( BB( 1 , 3 ) * AAA4 +
     .                                   BB( 2 , 3 ) * AAA5 +
     .                                   BB( 3 , 3 ) * AAA6 )
C
            PGRAD( IS , 1 ) = DTRMIN * ( BB( 1 , 4 ) * AAA1 +
     .                                   BB( 2 , 4 ) * AAA2 +
     .                                   BB( 3 , 4 ) * AAA3 )
C
            PGRAD( IS , 2 ) = DTRMIN * ( BB( 1 , 4 ) * AAA4 +
     .                                   BB( 2 , 4 ) * AAA5 +
     .                                   BB( 3 , 4 ) * AAA6 )
C
  105   CONTINUE
C
```

```
      NS1 = NS2 + 1
      NS2 = NS2 + NOFVES( INS + 1 )
   90 CONTINUE
C
C=================================================================
C
C --- CALL THE MONOTONICITY LIMITER --------------------------------
C
      CALL MONOTN
C
C=================================================================
C
C
C --- EXIT POINT FROM SUBROUTINE -----------------------------------
C
C     ------
      RETURN
C     ------
C
C     ---
      END
```

Thu Jul 1 14:15:55 1993 gradhd.f SUBROUTINE GRADNT

```
      SUBROUTINE GRADNT
C
C-----------------------------------------------------------I
C                                                           I
C      GRADNT COMPUTE THE GRADIENT FOR SECOND ORDER CALCULATION   I
C             USE THE INFORMATION IN THE THREE NEIGHBOURING       I
C             TRIANGLES THAT HAVE COMMON EDGES TO COMPUTE         I
C             GRADIENT APPLYING LEAST SQUARE TECHNIQUE            I
C                                                           I
C-----------------------------------------------------------I
C
      include    'cmsh00.h'
      include    'chyd00.h'
      include    'cint00.h'
      include    'cphs10.h'
      include    'cphs20.h'
C
C=================================================================
C
      REAL RRMIDL(MBP),PPMIDL(MBP),UUMIDL(MBP),VVMIDL(MBP)
      REAL RIGRAD(MBP),PIGRAD(MBP),UIGRAD(MBP),VIGRAD(MBP)
      REAL RJGRAD(MBP),PJGRAD(MBP),UJGRAD(MBP),VJGRAD(MBP)
      REAL RMAX(MBP),PMAX(MBP),UMAX(MBP),VMAX(MBP)
      REAL RMIN(MBP),PMIN(MBP),UMIN(MBP),VMIN(MBP)
      REAL RLEFTT(MBP),ULEFTT(MBP),VLEFTT(MBP),PLEFTT(MBP)
      REAL RRIGHT(MBP),URIGHT(MBP),VRIGHT(MBP),PRIGHT(MBP)
      REAL ROR(3),UOR(3),VOR(3),POR(3)
      REAL ROL(3),UOL(3),VOL(3),POL(3)
      REAL AA(3,3),BB(3,4),B(3),INDX(3),ATEMP(3,3,3),BTEMP(3,4,3)
      REAL AA0(3,3),BB0(3,4)
C
C=================================================================
C
C --- BEGIN LOOP OVER ALL CELLS IN THE DOMAIN ------------------------
C
      NS1 = 1
      NS2 = NOFVES( 1 )
      DO 90 INS = 1 , NVEES
C
C --- FETCH HYDRO QUANTITIES ---------------------------------------
C
      DO 105 IS = NS1 , NS2
C
       XSM = XS( 1 , IS )
       YSM = XS( 2 , IS )
       XSM2 = XSM * XSM
       YSM2 = YSM * YSM
       XYSM = XSM * YSM
C
       AA0( 1 , 1 ) = 1.0
```

```
         AAO( 1 , 2 ) = XSM
         AAO( 1 , 3 ) = YSM
C
         AAO( 2 , 1 ) = XSM
         AAO( 2 , 2 ) = XSM2
         AAO( 2 , 3 ) = XYSM
C
         AAO( 3 , 1 ) = YSM
         AAO( 3 , 2 ) = XYSM
         AAO( 3 , 3 ) = YSM2
C
         BB1 = HYDV( IS , 1 )
         BB2 = HYDV( IS , 2 )
         BB3 = HYDV( IS , 3 )
         BB4 = HYDV( IS , 4 )
C
         BB1X = BB1 * XSM
         BB2X = BB2 * XSM
         BB3X = BB3 * XSM
         BB4X = BB4 * XSM
C
         BB1Y = BB1 * YSM
         BB2Y = BB2 * YSM
         BB3Y = BB3 * YSM
         BB4Y = BB4 * YSM
C
         BBO( 1 , 1 ) = BB1
         BBO( 1 , 2 ) = BB2
         BBO( 1 , 3 ) = BB3
         BBO( 1 , 4 ) = BB4
C
         BBO( 2 , 1 ) = BB1X
         BBO( 2 , 2 ) = BB2X
         BBO( 2 , 3 ) = BB3X
         BBO( 2 , 4 ) = BB4X
C
         BBO( 3 , 1 ) = BB1Y
         BBO( 3 , 2 ) = BB2Y
         BBO( 3 , 3 ) = BB3Y
         BBO( 3 , 4 ) = BB4Y
C
         DO 115 IK = 1 , 3
         IE = JS( IK + 3 , IS )
         IF( IE . GT . 0 ) THEN
         ISS = JE( 4 , IE )
         ELSE
         ISS = JE( 3 , - IE )
         END IF
C
         IF( ISS . NE . 0 ) THEN
         XSS = XS( 1 , ISS )
         YSS = XS( 2 , ISS )
C
         HYDVR = HYDV( ISS , 1 )
         HYDVU = HYDV( ISS , 2 )
         HYDVV = HYDV( ISS , 3 )
         HYDVP = HYDV( ISS , 4 )
C
         ELSE
C
         IE = IABS( IE )
         HYDVR = BB1
         HYDVU = BB2
         HYDVV = BB3
         HYDVP = BB4
C
         XSS = 2. * XMIDL( IE ) - XSM
         YSS = 2. * YMIDL( IE ) - YSM
C
C         IJE5 = JE( 5 , IE )
C         IF( IJE5 . EQ . 6 . OR . IJE5 . EQ . 5 ) THEN
C         UUVV =   - ( BB2 * XN( IE ) + BB3 * YN( IE ) )
C         VVUU =     - BB2 * YN( IE ) + BB3 * XN( IE )
C         HYDVU = UUVV * XN( IE ) - VVUU * YN( IE )
```

```
C          HYDVV = UUVV * YN( IE ) + VVUU * XN( IE )
C
C          ELSEIF( IJE5 . EQ . 8 ) THEN
C          HYDVR =   RIN
C          HYDVU =   UIN
C          HYDVV =   VIN
C          HYDVP =   PIN
C
C          END IF
         END IF
C
         XSS2 = XSS * XSS
         YSS2 = YSS * YSS
         XYSS = XSS * YSS
C
         ATEMP( 1 , 1 , IK ) = 1.0
         ATEMP( 1 , 2 , IK ) = XSS
         ATEMP( 1 , 3 , IK ) = YSS
C
         ATEMP( 2 , 1 , IK ) = XSS
         ATEMP( 2 , 2 , IK ) = XSS2
         ATEMP( 2 , 3 , IK ) = XYSS
C
         ATEMP( 3 , 1 , IK ) = YSS
         ATEMP( 3 , 2 , IK ) = XYSS
         ATEMP( 3 , 3 , IK ) = YSS2
C
         BTEMP( 1 , 1 , IK ) = HYDVR
         BTEMP( 1 , 2 , IK ) = HYDVU
         BTEMP( 1 , 3 , IK ) = HYDVV
         BTEMP( 1 , 4 , IK ) = HYDVP
C
         BTEMP( 2 , 1 , IK ) = HYDVR * XSS
         BTEMP( 2 , 2 , IK ) = HYDVU * XSS
         BTEMP( 2 , 3 , IK ) = HYDVV * XSS
         BTEMP( 2 , 4 , IK ) = HYDVP * XSS
C
         BTEMP( 3 , 1 , IK ) = HYDVR * YSS
         BTEMP( 3 , 2 , IK ) = HYDVU * YSS
         BTEMP( 3 , 3 , IK ) = HYDVV * YSS
         BTEMP( 3 , 4 , IK ) = HYDVP * YSS
C
 115  CONTINUE
C
        AA( 1 , 1 ) = AA0( 1 , 1 ) * 3.  + ATEMP( 1 , 1 , 1 ) +
     .                ATEMP( 1 , 1 , 2 ) + ATEMP( 1 , 1 , 3 )
        AA( 1 , 2 ) = AA0( 1 , 2 ) * 3.  + ATEMP( 1 , 2 , 1 ) +
     .                ATEMP( 1 , 2 , 2 ) + ATEMP( 1 , 2 , 3 )
        AA( 1 , 3 ) = AA0( 1 , 3 ) * 3.  + ATEMP( 1 , 3 , 1 ) +
     .                ATEMP( 1 , 3 , 2 ) + ATEMP( 1 , 3 , 3 )
C
        AA( 2 , 1 ) = AA0( 2 , 1 ) * 3.  + ATEMP( 2 , 1 , 1 ) +
     .                ATEMP( 2 , 1 , 2 ) + ATEMP( 2 , 1 , 3 )
        AA( 2 , 2 ) = AA0( 2 , 2 ) * 3.  + ATEMP( 2 , 2 , 1 ) +
     .                ATEMP( 2 , 2 , 2 ) + ATEMP( 2 , 2 , 3 )
        AA( 2 , 3 ) = AA0( 2 , 3 ) * 3.  + ATEMP( 2 , 3 , 1 ) +
     .                ATEMP( 2 , 3 , 2 ) + ATEMP( 2 , 3 , 3 )
C
        AA( 3 , 1 ) = AA0( 3 , 1 ) * 3.  + ATEMP( 3 , 1 , 1 ) +
     .                ATEMP( 3 , 1 , 2 ) + ATEMP( 3 , 1 , 3 )
        AA( 3 , 2 ) = AA0( 3 , 2 ) * 3.  + ATEMP( 3 , 2 , 1 ) +
     .                ATEMP( 3 , 2 , 2 ) + ATEMP( 3 , 2 , 3 )
        AA( 3 , 3 ) = AA0( 3 , 3 ) * 3.  + ATEMP( 3 , 3 , 1 ) +
     .                ATEMP( 3 , 3 , 2 ) + ATEMP( 3 , 3 , 3 )
C
        BB( 1 , 1 ) = BB0( 1 , 1 ) * 3.  + BTEMP( 1 , 1 , 1 ) +
     .                BTEMP( 1 , 1 , 2 ) + BTEMP( 1 , 1 , 3 )
        BB( 1 , 2 ) = BB0( 1 , 2 ) * 3.  + BTEMP( 1 , 2 , 1 ) +
     .                BTEMP( 1 , 2 , 2 ) + BTEMP( 1 , 2 , 3 )
        BB( 1 , 3 ) = BB0( 1 , 3 ) * 3.  + BTEMP( 1 , 3 , 1 ) +
     .                BTEMP( 1 , 3 , 2 ) + BTEMP( 1 , 3 , 3 )
        BB( 1 , 4 ) = BB0( 1 , 4 )  * 3. + BTEMP( 1 , 4 , 1 ) +
     .                BTEMP( 1 , 4 , 2 ) + BTEMP( 1 , 4 , 3 )
C
```

```
         BB( 2 , 1 ) = BBO( 2 , 1 )  * 3. + BTEMP( 2 , 1 , 1 ) +
     .                 BTEMP( 2 , 1 , 2 ) + BTEMP( 2 , 1 , 3 )
         BB( 2 , 2 ) = BBO( 2 , 2 )  * 3. + BTEMP( 2 , 2 , 1 ) +
     .                 BTEMP( 2 , 2 , 2 ) + BTEMP( 2 , 2 , 3 )
         BB( 2 , 3 ) = BBO( 2 , 3 )  * 3. + BTEMP( 2 , 3 , 1 ) +
     .                 BTEMP( 2 , 3 , 2 ) + BTEMP( 2 , 3 , 3 )
         BB( 2 , 4 ) = BBO( 2 , 4 )  * 3. + BTEMP( 2 , 4 , 1 ) +
     .                 BTEMP( 2 , 4 , 2 ) + BTEMP( 2 , 4 , 3 )
C
         BB( 3 , 1 ) = BBO( 3 , 1 )  * 3. + BTEMP( 3 , 1 , 1 ) +
     .                 BTEMP( 3 , 1 , 2 ) + BTEMP( 3 , 1 , 3 )
         BB( 3 , 2 ) = BBO( 3 , 2 )  * 3. + BTEMP( 3 , 2 , 1 ) +
     .                 BTEMP( 3 , 2 , 2 ) + BTEMP( 3 , 2 , 3 )
         BB( 3 , 3 ) = BBO( 3 , 3 )  * 3. + BTEMP( 3 , 3 , 1 ) +
     .                 BTEMP( 3 , 3 , 2 ) + BTEMP( 3 , 3 , 3 )
         BB( 3 , 4 ) = BBO( 3 , 4 )  * 3. + BTEMP( 3 , 4 , 1 ) +
     .                 BTEMP( 3 , 4 , 2 ) + BTEMP( 3 , 4 , 3 )
C
         DETERM = AA( 1 , 1 ) * ( AA( 2 , 2 ) * AA( 3 , 3 ) -
     .                            AA( 3 , 2 ) * AA( 2 , 3 ) ) +
     .            AA( 2 , 1 ) * ( AA( 1 , 3 ) * AA( 3 , 2 ) -
     .                            AA( 3 , 3 ) * AA( 1 , 2 ) ) +
     .            AA( 3 , 1 ) * ( AA( 1 , 2 ) * AA( 2 , 3 ) -
     .                            AA( 2 , 2 ) * AA( 1 , 3 ) )
C
         DTRMIN = 1. / DETERM
C
         AAA1 = AA( 2 , 3 ) * AA( 3 , 1 ) - AA( 2 , 1 ) * AA( 3 , 3 )
         AAA2 = AA( 3 , 3 ) * AA( 1 , 1 ) - AA( 3 , 1 ) * AA( 1 , 3 )
         AAA3 = AA( 1 , 3 ) * AA( 2 , 1 ) - AA( 1 , 1 ) * AA( 2 , 3 )
C
         AAA4 = AA( 2 , 1 ) * AA( 3 , 2 ) - AA( 3 , 1 ) * AA( 2 , 2 )
         AAA5 = AA( 3 , 1 ) * AA( 1 , 2 ) - AA( 1 , 1 ) * AA( 3 , 2 )
         AAA6 = AA( 1 , 1 ) * AA( 2 , 2 ) - AA( 2 , 1 ) * AA( 1 , 2 )
C
         RGRAD( IS , 1 ) = DTRMIN * ( BB( 1 , 1 ) * AAA1 +
     .                                BB( 2 , 1 ) * AAA2 +
     .                                BB( 3 , 1 ) * AAA3 )
C
         RGRAD( IS , 2 ) = DTRMIN * ( BB( 1 , 1 ) * AAA4 +
     .                                BB( 2 , 1 ) * AAA5 +
     .                                BB( 3 , 1 ) * AAA6 )
C
         UGRAD( IS , 1 ) = DTRMIN * ( BB( 1 , 2 ) * AAA1 +
     .                                BB( 2 , 2 ) * AAA2 +
     .                                BB( 3 , 2 ) * AAA3 )
C
         UGRAD( IS , 2 ) = DTRMIN * ( BB( 1 , 2 ) * AAA4 +
     .                                BB( 2 , 2 ) * AAA5 +
     .                                BB( 3 , 2 ) * AAA6 )
C
         VGRAD( IS , 1 ) = DTRMIN * ( BB( 1 , 3 ) * AAA1 +
     .                                BB( 2 , 3 ) * AAA2 +
     .                                BB( 3 , 3 ) * AAA3 )
C
         VGRAD( IS , 2 ) = DTRMIN * ( BB( 1 , 3 ) * AAA4 +
     .                                BB( 2 , 3 ) * AAA5 +
     .                                BB( 3 , 3 ) * AAA6 )
C
         PGRAD( IS , 1 ) = DTRMIN * ( BB( 1 , 4 ) * AAA1 +
     .                                BB( 2 , 4 ) * AAA2 +
     .                                BB( 3 , 4 ) * AAA3 )
C
         PGRAD( IS , 2 ) = DTRMIN * ( BB( 1 , 4 ) * AAA4 +
     .                                BB( 2 , 4 ) * AAA5 +
     .                                BB( 3 , 4 ) * AAA6 )
C
  105  CONTINUE
C
       NS1 = NS2 + 1
       NS2 = NS2 + NOFVES( INS + 1 )
   90  CONTINUE
C
C=====================================================================
```

```
C
C --- CALL THE MONOTONICITY LIMITER ---------------------------------
C
      CALL MONOTN
C
C===================================================================
C
C
C --- EXIT POINT FROM SUBROUTINE ------------------------------------
C
C     ------
      RETURN
C     ------
C
C     ---
      END
```

Thu Jul 1 14:15:55 1993 gradhd.f SUBROUTINE MONOTN

```
      SUBROUTINE MONOTN
C
C-------------------------------------------------------------I
C                                                             I
C          MONOTN LIMIT THE GRADIENTS SO THAT NO NEW EXTREMUM ARE  I
C          CREATED ARTIFICIALY DURING THE PROJECTION PROCESS       I
C                                                             I
C-------------------------------------------------------------I
C
      include     'cmsh00.h'
      include     'chyd00.h'
      include     'cint00.h'
      include     'cphs10.h'
      include     'cphs20.h'
C
C===================================================================
C
      REAL RRMIDL(MBP),PPMIDL(MBP),UUMIDL(MBP),VVMIDL(MBP)
      REAL RIGRAD(MBP),PIGRAD(MBP),UIGRAD(MBP),VIGRAD(MBP)
      REAL RJGRAD(MBP),PJGRAD(MBP),UJGRAD(MBP),VJGRAD(MBP)
      REAL RMAX(MBP),PMAX(MBP),UMAX(MBP),VMAX(MBP)
      REAL RMIN(MBP),PMIN(MBP),UMIN(MBP),VMIN(MBP)
      REAL RLEFTT(MBP),ULEFTT(MBP),VLEFTT(MBP),PLEFTT(MBP)
      REAL RRIGHT(MBP),URIGHT(MBP),VRIGHT(MBP),PRIGHT(MBP)
      REAL ROR(3),UOR(3),VOR(3),POR(3)
      REAL ROL(3),UOL(3),VOL(3),POL(3)
      REAL AA(3,3),BB(3,4),B(3),INDX(3),ATEMP(3,3,3),BTEMP(3,4,3)
      REAL AA0(3,3),BB0(3,4)
C
C===================================================================
C
C --- LIMITER FOR GRADIENTS BEGINS ----------------------------------
C     USED TO PREVENT NEW MINIMA AND MAXIMA
C     AT PROJECTED INTERFACE VALUES.
C
      NS1 = 1
      NS2 = NOFVES( 1 )
      DO 80 INS = 1 , NVEES
C
      DO 150 IS = NS1 , NS2
       KS = IS - NS1 + 1
C
C --- FIRST TRIANGLE EDGE -------------------------------------------
C
       IE = IABS( JS( 4 , IS ) )
C
       ISL = JE( 3 , IE )
       ISR = JE( 4 , IE )
C
       RROL = HYDV( ISL , 1 )
       UUOL = HYDV( ISL , 2 )
       VVOL = HYDV( ISL , 3 )
       PPOL = HYDV( ISL , 4 )
C
       IJE5 = JE( 5 , IE )
```

```
      IF( IJE5 . EQ . 0 ) THEN
C
      RROR = HYDV( ISR , 1 )
      UUOR = HYDV( ISR , 2 )
      VVOR = HYDV( ISR , 3 )
      PPOR = HYDV( ISR , 4 )
C
      ELSE
C
      RROR = RROL
      UUOR = UUOL
      VVOR = VVOL
      PPOR = PPOL
C
C      IF( IJE5 . EQ . 6 . OR . IJE5 . EQ . 5 ) THEN
C      UUVV =   - ( UUOL * XN( IE ) + VVOL * YN( IE ) )
C      VVUU =     - UUOL * YN( IE ) + VVOL * XN( IE )
C      UUOR = UUVV * XN( IE ) - VVUU * YN( IE )
C      VVOR = UUVV * YN( IE ) + VVUU * XN( IE )
C
C      ELSE IF( IJE5 . EQ . 8 ) THEN
C      RROR =   RIN
C      UUOR =   UIN
C      VVOR =   VIN
C      PPOR =   PIN
C      END IF
C
      END IF
C
      ROL( 1 ) = RROL
      UOL( 1 ) = UUOL
      VOL( 1 ) = VVOL
      POL( 1 ) = PPOL
C
      ROR( 1 ) = RROR
      UOR( 1 ) = UUOR
      VOR( 1 ) = VVOR
      POR( 1 ) = PPOR
C
C --- SECOND TRIANGLE EDGE -----------------------------------------
C
      IE = IABS( JS( 5 , IS ) )
C
      ISL = JE( 3 , IE )
      ISR = JE( 4 , IE )
C
      RROL = HYDV( ISL , 1 )
      UUOL = HYDV( ISL , 2 )
      VVOL = HYDV( ISL , 3 )
      PPOL = HYDV( ISL , 4 )
C
      IJE5 = JE( 5 , IE )
      IF( IJE5 . EQ . 0 ) THEN
C
      RROR = HYDV( ISR , 1 )
      UUOR = HYDV( ISR , 2 )
      VVOR = HYDV( ISR , 3 )
      PPOR = HYDV( ISR , 4 )
C
      ELSE
C
      RROR = RROL
      UUOR = UUOL
      VVOR = VVOL
      PPOR = PPOL
C
C      IF( IJE5 . EQ . 6 . OR . IJE5 . EQ . 5 ) THEN
C      UUVV =   - ( UUOL * XN( IE ) + VVOL * YN( IE ) )
C      VVUU =     - UUOL * YN( IE ) + VVOL * XN( IE )
C      UUOR = UUVV * XN( IE ) - VVUU * YN( IE )
C      VVOR = UUVV * YN( IE ) + VVUU * XN( IE )
C
C      ELSE IF( IJE5 . EQ . 8 ) THEN
C      RROR =   RIN
```

```
C        UUOR =   UIN
C        VVOR =   VIN
C        PPOR =   PIN
C        END IF
C
       END IF
C
       ROL( 2 ) = RROL
       UOL( 2 ) = UUOL
       VOL( 2 ) = VVOL
       POL( 2 ) = PPOL
C
       ROR( 2 ) = RROR
       UOR( 2 ) = UUOR
       VOR( 2 ) = VVOR
       POR( 2 ) = PPOR
C
C --- THIRD TRIANGLE EDGE -------------------------------------------
C
       IE = IABS( JS( 6 , IS ) )
C
       ISL = JE( 3 , IE )
       ISR = JE( 4 , IE )
       RROL = HYDV( ISL , 1 )
       UUOL = HYDV( ISL , 2 )
       VVOL = HYDV( ISL , 3 )
       PPOL = HYDV( ISL , 4 )
C
       IJE5 = JE( 5 , IE )
       IF( IJE5 . EQ . 0 ) THEN
C
       RROR = HYDV( ISR , 1 )
       UUOR = HYDV( ISR , 2 )
       VVOR = HYDV( ISR , 3 )
       PPOR = HYDV( ISR , 4 )
C
       ELSE
C
       RROR = RROL
       UUOR = UUOL
       VVOR = VVOL
       PPOR = PPOL
C
C        IF( IJE5 . EQ . 6 . OR . IJE5 . EQ . 5 ) THEN
C        UUVV =    - ( UUOL * XN( IE ) + VVOL * YN( IE ) )
C        VVUU =      - UUOL * YN( IE ) + VVOL * XN( IE )
C        UUOR = UUVV * XN( IE ) - VVUU * YN( IE )
C        VVOR = UUVV * YN( IE ) + VVUU * XN( IE )
C
C        ELSE IF( IJE5 . EQ . 8 ) THEN
C        RROR =   RIN
C        UUOR =   UIN
C        VVOR =   VIN
C        PPOR =   PIN
C        END IF
C
       END IF
C
       ROL( 3 ) = RROL
       UOL( 3 ) = UUOL
       VOL( 3 ) = VVOL
       POL( 3 ) = PPOL
C
       ROR( 3 ) = RROR
       UOR( 3 ) = UUOR
       VOR( 3 ) = VVOR
       POR( 3 ) = PPOR
C
C --- FIND MAXIMA IN THE NEIGHBORHOOD OF A TRIANGLE -------------------
C
       RMAX( KS ) = AMAX1( ROL( 1 ) , ROL( 2 ) , ROL( 3 ) ,
     .                     ROR( 1 ) , ROR( 2 ) , ROR( 3 ) )
       UMAX( KS ) = AMAX1( UOL( 1 ) , UOL( 2 ) , UOL( 3 ) ,
     .                     UOR( 1 ) , UOR( 2 ) , UOR( 3 ) )
```

```
      VMAX( KS ) = AMAX1( VOL( 1 ) , VOL( 2 ) , VOL( 3 ) ,
     .                    VOR( 1 ) , VOR( 2 ) , VOR( 3 ) )
      PMAX( KS ) = AMAX1( POL( 1 ) , POL( 2 ) , POL( 3 ) ,
     .                    POR( 1 ) , POR( 2 ) , POR( 3 ) )
C
C --- FIND MINIMA IN THE NEIGHBORHOOD OF A TRIANGLE -------------------
C
      RMIN( KS ) = AMIN1( ROL( 1 ) , ROL( 2 ) , ROL( 3 ) ,
     .                    ROR( 1 ) , ROR( 2 ) , ROR( 3 ) )
      UMIN( KS ) = AMIN1( UOL( 1 ) , UOL( 2 ) , UOL( 3 ) ,
     .                    UOR( 1 ) , UOR( 2 ) , UOR( 3 ) )
      VMIN( KS ) = AMIN1( VOL( 1 ) , VOL( 2 ) , VOL( 3 ) ,
     .                    VOR( 1 ) , VOR( 2 ) , VOR( 3 ) )
      PMIN( KS ) = AMIN1( POL( 1 ) , POL( 2 ) , POL( 3 ) ,
     .                    POR( 1 ) , POR( 2 ) , POR( 3 ) )
C
 150  CONTINUE
C
C --- FIND DIFFERENCES BETWEEN EXTREMA AND THE TRIANGLE CENTERED -------
C     QUANTITIES
C
      DO 180 IS = NS1 , NS2
         KS = IS - NS1 + 1
C
         RRR( KS ) = RMAX( KS ) - HYDV( IS , 1 )
         RRL( KS ) = RMIN( KS ) - HYDV( IS , 1 )
         UUR( KS ) = UMAX( KS ) - HYDV( IS , 2 )
         UUL( KS ) = UMIN( KS ) - HYDV( IS , 2 )
         VVR( KS ) = VMAX( KS ) - HYDV( IS , 3 )
         VVL( KS ) = VMIN( KS ) - HYDV( IS , 3 )
         PPR( KS ) = PMAX( KS ) - HYDV( IS , 4 )
         PPL( KS ) = PMIN( KS ) - HYDV( IS , 4 )
C
 180  CONTINUE
C
C --- FIND THE PROJECTED INCRAMENTS FOR INTERFACE BASED QUANTITIES -----
C
      DO 170 IS = NS1 , NS2
         KS = IS - NS1 + 1
C
C --- FIRST TRIANGLE EDGE --------------------------------------------
C
        IE = IABS( JS( 4 , IS ) )
C
        ISL = JE( 3 , IE )
        ISR = JE( 4 , IE )
C
        XML = XMIDL( IE ) - XS( 1 , ISL )
        YML = YMIDL( IE ) - XS( 2 , ISL )
C
        RROL = 1.E-12 +
     .          RGRAD( ISL , 1 ) * XML + RGRAD( ISL , 2 ) * YML
        UUOL = 1.E-12 +
     .          UGRAD( ISL , 1 ) * XML + UGRAD( ISL , 2 ) * YML
        VVOL = 1.E-12 +
     .          VGRAD( ISL , 1 ) * XML + VGRAD( ISL , 2 ) * YML
        PPOL = 1.E-12 +
     .          PGRAD( ISL , 1 ) * XML + PGRAD( ISL , 2 ) * YML
C
        IJE5 = JE( 5 , IE )
        IF( IJE5 . EQ . 0 ) THEN
C
        XMR = XMIDL( IE ) - XS( 1 , ISR )
        YMR = YMIDL( IE ) - XS( 2 , ISR )
C
        RROR = 1.E-12 +
     .          RGRAD( ISR , 1 ) * XMR + RGRAD( ISR , 2 ) * YMR
        UUOR = 1.E-12 +
     .          UGRAD( ISR , 1 ) * XMR + UGRAD( ISR , 2 ) * YMR
        VVOR = 1.E-12 +
     .          VGRAD( ISR , 1 ) * XMR + VGRAD( ISR , 2 ) * YMR
        PPOR = 1.E-12 +
     .          PGRAD( ISR , 1 ) * XMR + PGRAD( ISR , 2 ) * YMR
C
```

```
      ELSE
C
      RROR = RROL
      UUOR = UUOL
      VVOR = VVOL
      PPOR = PPOL
C
      END IF
C
      ROL( 1 ) = 1. / RROL
      UOL( 1 ) = 1. / UUOL
      VOL( 1 ) = 1. / VVOL
      POL( 1 ) = 1. / PPOL
C
      ROR( 1 ) = 1. / RROR
      UOR( 1 ) = 1. / UUOR
      VOR( 1 ) = 1. / VVOR
      POR( 1 ) = 1. / PPOR
C
C --- SECOND TRIANGLE EDGE -------------------------------------------
C
      IE = IABS( JS( 5 , IS ) )
C
      ISL = JE( 3 , IE )
      ISR = JE( 4 , IE )
C
      XML = XMIDL( IE ) - XS( 1 , ISL )
      YML = YMIDL( IE ) - XS( 2 , ISL )
C
      RROL = 1.E-12 +
     .        RGRAD( ISL , 1 ) * XML + RGRAD( ISL , 2 ) * YML
      UUOL = 1.E-12 +
     .        UGRAD( ISL , 1 ) * XML + UGRAD( ISL , 2 ) * YML
      VVOL = 1.E-12 +
     .        VGRAD( ISL , 1 ) * XML + VGRAD( ISL , 2 ) * YML
      PPOL = 1.E-12 +
     .        PGRAD( ISL , 1 ) * XML + PGRAD( ISL , 2 ) * YML
C
      IJE5 = JE( 5 , IE )
      IF( IJE5 . EQ . 0 ) THEN
C
      XMR = XMIDL( IE ) - XS( 1 , ISR )
      YMR = YMIDL( IE ) - XS( 2 , ISR )
C
      RROR = 1.E-12 +
     .        RGRAD( ISR , 1 ) * XMR + RGRAD( ISR , 2 ) * YMR
      UUOR = 1.E-12 +
     .        UGRAD( ISR , 1 ) * XMR + UGRAD( ISR , 2 ) * YMR
      VVOR = 1.E-12 +
     .        VGRAD( ISR , 1 ) * XMR + VGRAD( ISR , 2 ) * YMR
      PPOR = 1.E-12 +
     .        PGRAD( ISR , 1 ) * XMR + PGRAD( ISR , 2 ) * YMR
C
      ELSE
C
      RROR = RROL
      UUOR = UUOL
      VVOR = VVOL
      PPOR = PPOL
C
      END IF
C
      ROL( 2 ) = 1. / RROL
      UOL( 2 ) = 1. / UUOL
      VOL( 2 ) = 1. / VVOL
      POL( 2 ) = 1. / PPOL
C
      ROR( 2 ) = 1. / RROR
      UOR( 2 ) = 1. / UUOR
      VOR( 2 ) = 1. / VVOR
      POR( 2 ) = 1. / PPOR
C
C --- THIRD TRIANGLE EDGE --------------------------------------------
C
```

```
      IE = IABS( JS( 6 , IS ) )
C
      ISL = JE( 3 , IE )
      ISR = JE( 4 , IE )
C
      XML = XMIDL( IE ) - XS( 1 , ISL )
      YML = YMIDL( IE ) - XS( 2 , ISL )
C
      RROL = 1.E-12 +
     .         RGRAD( ISL , 1 ) * XML + RGRAD( ISL , 2 ) * YML
      UUOL = 1.E-12 +
     .         UGRAD( ISL , 1 ) * XML + UGRAD( ISL , 2 ) * YML
      VVOL = 1.E-12 +
     .         VGRAD( ISL , 1 ) * XML + VGRAD( ISL , 2 ) * YML
      PPOL = 1.E-12 +
     .         PGRAD( ISL , 1 ) * XML + PGRAD( ISL , 2 ) * YML
C
      IJE5 = JE( 5 , IE )
      IF( IJE5 . EQ . 0 ) THEN
C
      XMR = XMIDL( IE ) - XS( 1 , ISR )
      YMR = YMIDL( IE ) - XS( 2 , ISR )
C
      RROR = 1.E-12 +
     .         RGRAD( ISR , 1 ) * XMR + RGRAD( ISR , 2 ) * YMR
      UUOR = 1.E-12 +
     .         UGRAD( ISR , 1 ) * XMR + UGRAD( ISR , 2 ) * YMR
      VVOR = 1.E-12 +
     .         VGRAD( ISR , 1 ) * XMR + VGRAD( ISR , 2 ) * YMR
      PPOR = 1.E-12 +
     .         PGRAD( ISR , 1 ) * XMR + PGRAD( ISR , 2 ) * YMR
C
      ELSE
C
      RROR = RROL
      UUOR = UUOL
      VVOR = VVOL
      PPOR = PPOL
C
      END IF
C
      ROL( 3 ) = 1. / RROL
      UOL( 3 ) = 1. / UUOL
      VOL( 3 ) = 1. / VVOL
      POL( 3 ) = 1. / PPOL
C
      ROR( 3 ) = 1. / RROR
      UOR( 3 ) = 1. / UUOR
      VOR( 3 ) = 1. / VVOR
      POR( 3 ) = 1. / PPOR
C
      ISNR = SIGN( 1. , ROR( 1 ) )
      ISNL = SIGN( 1. , ROL( 1 ) )
C
C --- PERFORM THE LIMITING ON THE INCRAMENTS --------------------------
C
      TEMPR = ( 1 + ISNR ) * RRR( KS ) +
     .        ( 1 - ISNR ) * RRL( KS )
      RUVPR1 = 0.5 * TEMPR * ROR( 1 )
C
      TEMPL = ( 1 + ISNL ) * RRR( KS ) +
     .        ( 1 - ISNL ) * RRL( KS )
      RUVPL1 = 0.5 * TEMPL * ROL( 1 )
C
      ISNR = SIGN( 1. , ROR( 2 ) )
      ISNL = SIGN( 1. , ROL( 2 ) )
C
      TEMPR = ( 1 + ISNR ) * RRR( KS ) +
     .        ( 1 - ISNR ) * RRL( KS )
      RUVPR2 = 0.5 * TEMPR * ROR( 2 )
C
      TEMPL = ( 1 + ISNL ) * RRR( KS ) +
     .        ( 1 - ISNL ) * RRL( KS )
      RUVPL2 = 0.5 * TEMPL * ROL( 2 )
```

```
C
      ISNR = SIGN( 1. , ROR( 3 ) )
      ISNL = SIGN( 1. , ROL( 3 ) )
C
      TEMPR = ( 1 + ISNR ) * RRR( KS ) +
     .          ( 1 - ISNR ) * RRL( KS )
      RUVPR3 = 0.5 * TEMPR * ROR( 3 )
C
      TEMPL = ( 1 + ISNL ) * RRR( KS ) +
     .          ( 1 - ISNL ) * RRL( KS )
      RUVPL3 = 0.5 * TEMPL * ROL( 3 )
C
      RMIN( KS ) = AMIN1( 1. , RUVPR1 , RUVPL1 , RUVPR2 , RUVPL2 ,
     .                                           RUVPR3 , RUVPL3 )
C
      ISNR = SIGN( 1. , UOR( 1 ) )
      ISNL = SIGN( 1. , UOL( 1 ) )
C
      TEMPR = ( 1 + ISNR ) * UUR( KS ) +
     .          ( 1 - ISNR ) * UUL( KS )
      RUVPR1 = 0.5 * TEMPR * UOR( 1 )
C
      TEMPL = ( 1 + ISNL ) * UUR( KS ) +
     .          ( 1 - ISNL ) * UUL( KS )
      RUVPL1 = 0.5 * TEMPL * UOL( 1 )
C
      ISNR = SIGN( 1. , UOR( 2 ) )
      ISNL = SIGN( 1. , UOL( 2 ) )
C
      TEMPR = ( 1 + ISNR ) * UUR( KS ) +
     .          ( 1 - ISNR ) * UUL( KS )
      RUVPR2 = 0.5 * TEMPR * UOR( 2 )
C
      TEMPL = ( 1 + ISNL ) * UUR( KS ) +
     .          ( 1 - ISNL ) * UUL( KS )
      RUVPL2 = 0.5 * TEMPL * UOL( 2 )
C
      ISNR = SIGN( 1. , UOR( 3 ) )
      ISNL = SIGN( 1. , UOL( 3 ) )
C
      TEMPR = ( 1 + ISNR ) * UUR( KS ) +
     .          ( 1 - ISNR ) * UUL( KS )
      RUVPR3 = 0.5 * TEMPR * UOR( 3 )
C
      TEMPL = ( 1 + ISNL ) * UUR( KS ) +
     .          ( 1 - ISNL ) * UUL( KS )
      RUVPL3 = 0.5 * TEMPL * UOL( 3 )
C
      UMIN( KS ) = AMIN1( 1. , RUVPR1 , RUVPL1 , RUVPR2 , RUVPL2 ,
     .                                           RUVPR3 , RUVPL3 )
C
      ISNR = SIGN( 1. , VOR( 1 ) )
      ISNL = SIGN( 1. , VOL( 1 ) )
C
      TEMPR = ( 1 + ISNR ) * VVR( KS ) +
     .          ( 1 - ISNR ) * VVL( KS )
      RUVPR1 = 0.5 * TEMPR * VOR( 1 )
C
      TEMPL = ( 1 + ISNL ) * VVR( KS ) +
     .          ( 1 - ISNL ) * VVL( KS )
      RUVPL1 = 0.5 * TEMPL * VOL( 1 )
C
      ISNR = SIGN( 1. , VOR( 2 ) )
      ISNL = SIGN( 1. , VOL( 2 ) )
C
      TEMPR = ( 1 + ISNR ) * VVR( KS ) +
     .          ( 1 - ISNR ) * VVL( KS )
      RUVPR2 = 0.5 * TEMPR * VOR( 2 )
C
      TEMPL = ( 1 + ISNL ) * VVR( KS ) +
     .          ( 1 - ISNL ) * VVL( KS )
      RUVPL2 = 0.5 * TEMPL * VOL( 2 )
C
      ISNR = SIGN( 1. , VOR( 3 ) )
```

```
            ISNL = SIGN( 1. , VOL( 3 ) )
C
            TEMPR = ( 1 + ISNR ) * VVR( KS ) +
     .              ( 1 - ISNR ) * VVL( KS )
            RUVPR3 = 0.5 * TEMPR * VOR( 3 )
C
            TEMPL = ( 1 + ISNL ) * VVR( KS ) +
     .              ( 1 - ISNL ) * VVL( KS )
            RUVPL3 = 0.5 * TEMPL * VOL( 3 )
C
            VMIN( KS ) = AMIN1( 1. , RUVPR1 , RUVPL1 , RUVPR2 , RUVPL2 ,
     .                                           RUVPR3 , RUVPL3 )
C
            ISNR = SIGN( 1. , POR( 1 ) )
            ISNL = SIGN( 1. , POL( 1 ) )
C
            TEMPR = ( 1 + ISNR ) * PPR( KS ) +
     .              ( 1 - ISNR ) * PPL( KS )
            RUVPR1 = 0.5 * TEMPR * POR( 1 )
C
            TEMPL = ( 1 + ISNL ) * PPR( KS ) +
     .              ( 1 - ISNL ) * PPL( KS )
            RUVPL1 = 0.5 * TEMPL * POL( 1 )
C
            ISNR = SIGN( 1. , POR( 2 ) )
            ISNL = SIGN( 1. , POL( 2 ) )
C
            TEMPR = ( 1 + ISNR ) * PPR( KS ) +
     .              ( 1 - ISNR ) * PPL( KS )
            RUVPR2 = 0.5 * TEMPR * POR( 2 )
C
            TEMPL = ( 1 + ISNL ) * PPR( KS ) +
     .              ( 1 - ISNL ) * PPL( KS )
            RUVPL2 = 0.5 * TEMPL * POL( 2 )
C
            ISNR = SIGN( 1. , POR( 3 ) )
            ISNL = SIGN( 1. , POL( 3 ) )
C
            TEMPR = ( 1 + ISNR ) * PPR( KS ) +
     .              ( 1 - ISNR ) * PPL( KS )
            RUVPR3 = 0.5 * TEMPR * POR( 3 )
C
            TEMPL = ( 1 + ISNL ) * PPR( KS ) +
     .              ( 1 - ISNL ) * PPL( KS )
            RUVPL3 = 0.5 * TEMPL * POL( 3 )
C
            PMIN( KS ) = AMIN1( 1. , RUVPR1 , RUVPL1 , RUVPR2 , RUVPL2 ,
     .                                           RUVPR3 , RUVPL3 )
C
  170   CONTINUE
C
C --- LIMIT THE ACTUAL GRADIENTS ---------------------------------------
C
         DO 330 IH = 1 , 2
C
         DO 330 IS = NS1 , NS2
            KS = IS - NS1 + 1
C
          RGRAD( IS , IH ) = RGRAD( IS , IH ) * RMIN( KS ) * FLATDR
          UGRAD( IS , IH ) = UGRAD( IS , IH ) * UMIN( KS ) * FLATDR
          VGRAD( IS , IH ) = VGRAD( IS , IH ) * VMIN( KS ) * FLATDR
          PGRAD( IS , IH ) = PGRAD( IS , IH ) * PMIN( KS ) * FLATDR
C
  330   CONTINUE
C
         NS1 = NS2 + 1
         NS2 = NS2 + NOFVES( INS + 1 )
   80   CONTINUE
C
C======================================================================
C
C --- CALL THE CHARECTERISTIC LIMITER ---------------------------------
C
         CALL FCHART
```

```
C
C=================================================================
C
C
C --- EXIT POINT FROM SUBROUTINE -------------------------------------
C
C     ------
      RETURN
C     ------
C
C     ---
      END
```

Thu Jul 1 14:15:55 1993   gradhd.f   SUBROUTINE GRADNG

```
      SUBROUTINE GRADNG
C
C--------------------------------------------------------------I
C                                                              I
C      GRADNG COMPUTE THE GRADIENT FOR SECOND ORDER CALCULATION   I
C             USING THE INFORMATION STORED ASSOCIATED WITH THE    I
C             VERTICIES OF THE TRIANGLE TO COMPUTE THE GRADIENT   I
C                                                              I
C--------------------------------------------------------------I
C
      include     'cmsh00.h'
      include     'chyd00.h'
      include     'cint00.h'
      include     'cphs10.h'
      include     'cphs20.h'
C
C=================================================================
C
C --- BEGIN LOOP OVER ALL CELLS IN THE DOMAIN ------------------------
C
      NS1 = 1
      NS2 = NOFVES( 1 )
      DO 90 INS = 1 , NVEES
C
C --- FETCH HYDRO QUANTITIES ------------------------------------------
C
      DO 105 IS = NS1 , NS2
         KS = IS - NS1 + 1
C
       IV1 = JS( 1 , IS )
       IV2 = JS( 2 , IS )
       IV3 = JS( 3 , IS )
       XV1 = XV( 1 , IV1 )
       XV2 = XV( 1 , IV2 )
       XV3 = XV( 1 , IV3 )
       YV1 = XV( 2 , IV1 )
       YV2 = XV( 2 , IV2 )
       YV3 = XV( 2 , IV3 )
       C = ( XV2 - XV1 ) * ( YV3 - YV2 ) - ( XV3 - XV2 ) * ( YV2 - YV1 )
       CINV = 1. / C
C
       RRMDL1 = HYDVVV( IV1 , 1 )
       UUMDL1 = HYDVVV( IV1 , 2 ) / RRMDL1
       VVMDL1 = HYDVVV( IV1 , 3 ) / RRMDL1
       PPMDL1 = ( HYDVVV( IV1 , 4 ) - .5 * RRMDL1 * ( UUMDL1 * UUMDL1 +
     .                VVMDL1 * VVMDL1 ) ) * ( HYDVVV( IV1 , 5 ) - 1. )
C
       RRMDL2 = HYDVVV( IV2 , 1 )
       UUMDL2 = HYDVVV( IV2 , 2 ) / RRMDL2
       VVMDL2 = HYDVVV( IV2 , 3 ) / RRMDL2
       PPMDL2 = ( HYDVVV( IV2 , 4 ) - .5 * RRMDL2 * ( UUMDL2 * UUMDL2 +
     .                VVMDL2 * VVMDL2 ) ) * ( HYDVVV( IV2 , 5 ) - 1. )
C
       RRMDL3 = HYDVVV( IV3 , 1 )
       UUMDL3 = HYDVVV( IV3 , 2 ) / RRMDL3
       VVMDL3 = HYDVVV( IV3 , 3 ) / RRMDL3
       PPMDL3 = ( HYDVVV( IV3 , 4 ) - .5 * RRMDL3 * ( UUMDL3 * UUMDL3 +
     .                VVMDL3 * VVMDL3 ) ) * ( HYDVVV( IV3 , 5 ) - 1. )
C
```

```
            ZV1 = RRMDL1
            ZV2 = RRMDL2
            ZV3 = RRMDL3
            A = ( YV2 - YV1 ) * ( ZV3 - ZV2 ) - ( YV3 - YV2 ) * ( ZV2 - ZV1 )
            B = ( ZV2 - ZV1 ) * ( XV3 - XV2 ) - ( ZV3 - ZV2 ) * ( XV2 - XV1 )
C
            RGRAD( IS , 1 ) = - A * CINV
            RGRAD( IS , 2 ) = - B * CINV
C
            ZV1 = UUMDL1
            ZV2 = UUMDL2
            ZV3 = UUMDL3
            A = ( YV2 - YV1 ) * ( ZV3 - ZV2 ) - ( YV3 - YV2 ) * ( ZV2 - ZV1 )
            B = ( ZV2 - ZV1 ) * ( XV3 - XV2 ) - ( ZV3 - ZV2 ) * ( XV2 - XV1 )
C
            UGRAD( IS , 1 ) = - A * CINV
            UGRAD( IS , 2 ) = - B * CINV
C
            ZV1 = VVMDL1
            ZV2 = VVMDL2
            ZV3 = VVMDL3
            A = ( YV2 - YV1 ) * ( ZV3 - ZV2 ) - ( YV3 - YV2 ) * ( ZV2 - ZV1 )
            B = ( ZV2 - ZV1 ) * ( XV3 - XV2 ) - ( ZV3 - ZV2 ) * ( XV2 - XV1 )
C
            VGRAD( IS , 1 ) = - A * CINV
            VGRAD( IS , 2 ) = - B * CINV
C
            ZV1 = PPMDL1
            ZV2 = PPMDL2
            ZV3 = PPMDL3
            A = ( YV2 - YV1 ) * ( ZV3 - ZV2 ) - ( YV3 - YV2 ) * ( ZV2 - ZV1 )
            B = ( ZV2 - ZV1 ) * ( XV3 - XV2 ) - ( ZV3 - ZV2 ) * ( XV2 - XV1 )
C
            PGRAD( IS , 1 ) = - A * CINV
            PGRAD( IS , 2 ) = - B * CINV
C
105   CONTINUE
C
      NS1 = NS2 + 1
      NS2 = NS2 + NOFVES( INS + 1 )
 90   CONTINUE
C
C=====================================================================
C
C --- CALL THE MONOTONICITY LIMITER ---------------------------------
C
      CALL MONOTN
C
C=====================================================================
C
C
C --- EXIT POINT FROM SUBROUTINE -----------------------------------
C
C     ------
      RETURN
C     ------
C
C     ---
      END
```

```
      SUBROUTINE GRADNO
C
C------------------------------------------------------------I
C                                                            I
C      GRADNO COMPUTE THE GRADIENT FOR SECOND ORDER CALCULATION   I
C             USING THE INFORMATION STORED ASSOCIATED WITH THE    I
C             VERTICIES OF THE TRIANGLE TO COMPUTE THE GRADIENT   I
C             APPLYING THE GRADIENT THEOREM                       I
C                                                            I
C------------------------------------------------------------I
C
      include     'cmsh00.h'
      include     'chyd00.h'
      include     'cint00.h'
      include     'cphs10.h'
      include     'cphs20.h'
C
C==================================================================
C
      REAL RRMIDL(MBP),PPMIDL(MBP),UUMIDL(MBP),VVMIDL(MBP)
      REAL RIGRAD(MBP),PIGRAD(MBP),UIGRAD(MBP),VIGRAD(MBP)
      REAL RJGRAD(MBP),PJGRAD(MBP),UJGRAD(MBP),VJGRAD(MBP)
      REAL RMAX(MBP),PMAX(MBP),UMAX(MBP),VMAX(MBP)
      REAL RMIN(MBP),PMIN(MBP),UMIN(MBP),VMIN(MBP)
      REAL RLEFTT(MBP),ULEFTT(MBP),VLEFTT(MBP),PLEFTT(MBP)
      REAL RRIGHT(MBP),URIGHT(MBP),VRIGHT(MBP),PRIGHT(MBP)
      REAL ROR(3),UOR(3),VOR(3),POR(3)
      REAL ROL(3),UOL(3),VOL(3),POL(3)
      REAL AA(3,3),BB(3,4),B(3),INDX(3),ATEMP(3,3,3),BTEMP(3,4,3)
      REAL AA0(3,3),BB0(3,4)
C
C==================================================================
C
C --- BEGIN LOOP OVER ALL CELLS IN THE DOMAIN -------------------------
C
      DO 120 IH = 1 , 2
      DO 120 IS = 1 , NS
      RGRAD( IS , IH ) = 0.
      UGRAD( IS , IH ) = 0.
      VGRAD( IS , IH ) = 0.
      PGRAD( IS , IH ) = 0.
120   CONTINUE
C
      NE1 = 1
      NE2 = NOFVEE( 1 )
      DO 90 INE = 1 , NVEEE
C
C --- FETCH HYDRO QUANTITIES ----------------------------------------
C
      DO 105 IE = NE1 , NE2
         KE = IE - NE1 + 1
C
       IV1 = JE( 1 , IE )
       IV2 = JE( 2 , IE )
       RRMDL = ( HYDVVV( IV1 , 1 ) + HYDVVV( IV2 , 1 ) ) * .5
       UUMDL = ( HYDVVV( IV1 , 2 ) + HYDVVV( IV2 , 2 ) ) * .5 / RRMDL
       VVMDL = ( HYDVVV( IV1 , 3 ) + HYDVVV( IV2 , 3 ) ) * .5 / RRMDL
       PPMDL = ( HYDVVV( IV1 , 4 ) + HYDVVV( IV2 , 4 ) ) * .5
       GGMDL = ( HYDVVV( IV1 , 5 ) + HYDVVV( IV2 , 5 ) ) * .5
       PPMDL = ( PPMDL - .5 * RRMDL *
     .         ( UUMDL * UUMDL + VVMDL * VVMDL ) ) * ( GGMDL - 1. )
C
       RRMIDL( KE ) = RRMDL
       UUMIDL( KE ) = UUMDL
       VVMIDL( KE ) = VVMDL
       PPMIDL( KE ) = PPMDL
C
105   CONTINUE
C
      DO 110 IE = NE1 , NE2
         KE = IE - NE1 + 1
C
      XEXN = XE( 1 , IE ) * XN( IE )
      XEYN = XE( 1 , IE ) * YN( IE )
```

```
C
      RIGRAD( KE ) = RRMIDL( KE ) * XEXN
      UIGRAD( KE ) = UUMIDL( KE ) * XEXN
      VIGRAD( KE ) = VVMIDL( KE ) * XEXN
      PIGRAD( KE ) = PPMIDL( KE ) * XEXN
C
      RJGRAD( KE ) = RRMIDL( KE ) * XEYN
      UJGRAD( KE ) = UUMIDL( KE ) * XEYN
      VJGRAD( KE ) = VVMIDL( KE ) * XEYN
      PJGRAD( KE ) = PPMIDL( KE ) * XEYN
C
  110 CONTINUE
C
      DO 130 IE = NE1 , NE2
         KE = IE - NE1 + 1
C
       ISL = JE( 3 , IE )
       ISR = JE( 4 , IE )
       IJE5 = JE( 5 , IE )
C
       IF( IJE5 . EQ . 0 ) THEN
C
      RGRAD( ISL , 1 ) = RGRAD( ISL , 1 ) + RIGRAD( KE )
      RGRAD( ISR , 1 ) = RGRAD( ISR , 1 ) - RIGRAD( KE )
      RGRAD( ISL , 2 ) = RGRAD( ISL , 2 ) + RJGRAD( KE )
      RGRAD( ISR , 2 ) = RGRAD( ISR , 2 ) - RJGRAD( KE )
      UGRAD( ISL , 1 ) = UGRAD( ISL , 1 ) + UIGRAD( KE )
      UGRAD( ISR , 1 ) = UGRAD( ISR , 1 ) - UIGRAD( KE )
      UGRAD( ISL , 2 ) = UGRAD( ISL , 2 ) + UJGRAD( KE )
      UGRAD( ISR , 2 ) = UGRAD( ISR , 2 ) - UJGRAD( KE )
      VGRAD( ISL , 1 ) = VGRAD( ISL , 1 ) + VIGRAD( KE )
      VGRAD( ISR , 1 ) = VGRAD( ISR , 1 ) - VIGRAD( KE )
      VGRAD( ISL , 2 ) = VGRAD( ISL , 2 ) + VJGRAD( KE )
      VGRAD( ISR , 2 ) = VGRAD( ISR , 2 ) - VJGRAD( KE )
      PGRAD( ISL , 1 ) = PGRAD( ISL , 1 ) + PIGRAD( KE )
      PGRAD( ISR , 1 ) = PGRAD( ISR , 1 ) - PIGRAD( KE )
      PGRAD( ISL , 2 ) = PGRAD( ISL , 2 ) + PJGRAD( KE )
      PGRAD( ISR , 2 ) = PGRAD( ISR , 2 ) - PJGRAD( KE )
C
      ELSE
C
      RGRAD( ISL , 1 ) = RGRAD( ISL , 1 ) + RIGRAD( KE )
      RGRAD( ISL , 2 ) = RGRAD( ISL , 2 ) + RJGRAD( KE )
      UGRAD( ISL , 1 ) = UGRAD( ISL , 1 ) + UIGRAD( KE )
      UGRAD( ISL , 2 ) = UGRAD( ISL , 2 ) + UJGRAD( KE )
      VGRAD( ISL , 1 ) = VGRAD( ISL , 1 ) + VIGRAD( KE )
      VGRAD( ISL , 2 ) = VGRAD( ISL , 2 ) + VJGRAD( KE )
      PGRAD( ISL , 1 ) = PGRAD( ISL , 1 ) + PIGRAD( KE )
      PGRAD( ISL , 2 ) = PGRAD( ISL , 2 ) + PJGRAD( KE )
C
      END IF
C
  130 CONTINUE
      NE1 = NE2 + 1
      NE2 = NE2 + NOFVEE( INE + 1 )
   90 CONTINUE
C
      DO 140 IH = 1 , 2
      DO 140 IS = 1 , NS
      RGRAD( IS , IH ) = RGRAD( IS , IH ) * SAREA( IS )
      UGRAD( IS , IH ) = UGRAD( IS , IH ) * SAREA( IS )
      VGRAD( IS , IH ) = VGRAD( IS , IH ) * SAREA( IS )
      PGRAD( IS , IH ) = PGRAD( IS , IH ) * SAREA( IS )
  140 CONTINUE
C
C======================================================================
C
C --- CALL THE MONOTONICITY LIMITER ---------------------------------
C
      CALL MONOTN
C
C======================================================================
C
C
```

```
C --- EXIT POINT FROM SUBROUTINE ------------------------------------
C
C     ------
      RETURN
C     ------
C
C     ---
      END
```

Thu Jul 1 14:15:55 1993 gradhd.f SUBROUTINE GRADNS

```
      SUBROUTINE GRADNS
C
C------------------------------------------------------------------I
C                                                                  I
C       GRADNS COMPUTE THE GRADIENT FOR SECOND ORDER CALCULATION   I
C              USING THE INFORMATION ASSOCIATE WITH THE BARICENTER I
C              OF THE TWO TRIANGLES FROM THE TWO SIDE OF EACH      |
C              EDGE COMPUTING THE VALUE FOR THE EDGE AND APPLYING  I
C              THE GRADIENT THEOREM TO COMPUTE THE GRADIENT        I
C                                                                  I
C------------------------------------------------------------------I
C
      include    'cmsh00.h'
      include    'chyd00.h'
      include    'cint00.h'
      include    'cphs10.h'
      include    'cphs20.h'
C
C=====================================================================
C
      REAL RRMIDL(MBP),PPMIDL(MBP),UUMIDL(MBP),VVMIDL(MBP)
      REAL RIGRAD(MBP),PIGRAD(MBP),UIGRAD(MBP),VIGRAD(MBP)
      REAL RJGRAD(MBP),PJGRAD(MBP),UJGRAD(MBP),VJGRAD(MBP)
      REAL RMAX(MBP),PMAX(MBP),UMAX(MBP),VMAX(MBP)
      REAL RMIN(MBP),PMIN(MBP),UMIN(MBP),VMIN(MBP)
      REAL RLEFTT(MBP),ULEFTT(MBP),VLEFTT(MBP),PLEFTT(MBP)
      REAL RRIGHT(MBP),URIGHT(MBP),VRIGHT(MBP),PRIGHT(MBP)
      REAL ROR(3),UOR(3),VOR(3),POR(3)
      REAL ROL(3),UOL(3),VOL(3),POL(3)
      REAL AA(3,3),BB(3,4),B(3),INDX(3),ATEMP(3,3,3),BTEMP(3,4,3)
      REAL AA0(3,3),BB0(3,4)
C
C=====================================================================
C
C --- BEGIN LOOP OVER ALL CELLS IN THE DOMAIN ------------------------
C
      DO 120 IH = 1 , 2
      DO 120 IS = 1 , NS
      RGRAD( IS , IH ) = 0.
      UGRAD( IS , IH ) = 0.
      VGRAD( IS , IH ) = 0.
      PGRAD( IS , IH ) = 0.
  120 CONTINUE
C
      NE1 = 1
      NE2 = NOFVEE( 1 )
      DO 90 INE = 1 , NVEEE
C
C --- FETCH HYDRO QUANTITIES -----------------------------------------
C
      DO 105 IE = NE1 , NE2
         KE = IE - NE1 + 1
C
       ISL = JE( 3 , IE )
       ISR = JE( 4 , IE )
       IJE5 = JE( 5 , IE )
C
       IF( IJE5 . EQ . 0 ) THEN

       RRMDL = XYMIDL( IE ) * ( HYDV( ISR , 1 ) -
     .                          HYDV( ISL , 1 ) ) + HYDV( ISL , 1 )
       UUMDL = XYMIDL( IE ) * ( HYDV( ISR , 2 ) -
     .                          HYDV( ISL , 2 ) ) + HYDV( ISL , 2 )
```

```
            VVMDL = XYMIDL( IE ) * ( HYDV( ISR , 3 ) -
     .                               HYDV( ISL , 3 ) ) + HYDV( ISL , 3 )
            PPMDL = XYMIDL( IE ) * ( HYDV( ISR , 4 ) -
     .                               HYDV( ISL , 4 ) ) + HYDV( ISL , 4 )
C
            ELSE
C
            RRMDL = HYDV( ISL , 1 )
            UUMDL = HYDV( ISL , 2 )
            VVMDL = HYDV( ISL , 3 )
            PPMDL = HYDV( ISL , 4 )
C
            END IF
C
            RRMIDL( KE ) = RRMDL
            UUMIDL( KE ) = UUMDL
            VVMIDL( KE ) = VVMDL
            PPMIDL( KE ) = PPMDL
C
  105   CONTINUE
C
        DO 110 IE = NE1 , NE2
           KE = IE - NE1 + 1
C
        XEXN = XE( 1 , IE ) * XN( IE )
        XEYN = XE( 1 , IE ) * YN( IE )
C
        RIGRAD( KE ) = RRMIDL( KE ) * XEXN
        UIGRAD( KE ) = UUMIDL( KE ) * XEXN
        VIGRAD( KE ) = VVMIDL( KE ) * XEXN
        PIGRAD( KE ) = PPMIDL( KE ) * XEXN
C
        RJGRAD( KE ) = RRMIDL( KE ) * XEYN
        UJGRAD( KE ) = UUMIDL( KE ) * XEYN
        VJGRAD( KE ) = VVMIDL( KE ) * XEYN
        PJGRAD( KE ) = PPMIDL( KE ) * XEYN
C
  110   CONTINUE
C
        DO 130 IE = NE1 , NE2
           KE = IE - NE1 + 1
C
         ISL = JE( 3 , IE )
         ISR = JE( 4 , IE )
         IJE5 = JE( 5 , IE )
C
         IF( IJE5 . EQ . 0 ) THEN
C
        RGRAD( ISL , 1 ) = RGRAD( ISL , 1 ) + RIGRAD( KE )
        RGRAD( ISR , 1 ) = RGRAD( ISR , 1 ) - RIGRAD( KE )
        RGRAD( ISL , 2 ) = RGRAD( ISL , 2 ) + RJGRAD( KE )
        RGRAD( ISR , 2 ) = RGRAD( ISR , 2 ) - RJGRAD( KE )
        UGRAD( ISL , 1 ) = UGRAD( ISL , 1 ) + UIGRAD( KE )
        UGRAD( ISR , 1 ) = UGRAD( ISR , 1 ) - UIGRAD( KE )
        UGRAD( ISL , 2 ) = UGRAD( ISL , 2 ) + UJGRAD( KE )
        UGRAD( ISR , 2 ) = UGRAD( ISR , 2 ) - UJGRAD( KE )
        VGRAD( ISL , 1 ) = VGRAD( ISL , 1 ) + VIGRAD( KE )
        VGRAD( ISR , 1 ) = VGRAD( ISR , 1 ) - VIGRAD( KE )
        VGRAD( ISL , 2 ) = VGRAD( ISL , 2 ) + VJGRAD( KE )
        VGRAD( ISR , 2 ) = VGRAD( ISR , 2 ) - VJGRAD( KE )
        PGRAD( ISL , 1 ) = PGRAD( ISL , 1 ) + PIGRAD( KE )
        PGRAD( ISR , 1 ) = PGRAD( ISR , 1 ) - PIGRAD( KE )
        PGRAD( ISL , 2 ) = PGRAD( ISL , 2 ) + PJGRAD( KE )
        PGRAD( ISR , 2 ) = PGRAD( ISR , 2 ) - PJGRAD( KE )
C
        ELSE
C
        RGRAD( ISL , 1 ) = RGRAD( ISL , 1 ) + RIGRAD( KE )
        RGRAD( ISL , 2 ) = RGRAD( ISL , 2 ) + RJGRAD( KE )
        UGRAD( ISL , 1 ) = UGRAD( ISL , 1 ) + UIGRAD( KE )
        UGRAD( ISL , 2 ) = UGRAD( ISL , 2 ) + UJGRAD( KE )
        VGRAD( ISL , 1 ) = VGRAD( ISL , 1 ) + VIGRAD( KE )
        VGRAD( ISL , 2 ) = VGRAD( ISL , 2 ) + VJGRAD( KE )
        PGRAD( ISL , 1 ) = PGRAD( ISL , 1 ) + PIGRAD( KE )
```

```
      PGRAD( ISL , 2 ) = PGRAD( ISL , 2 ) + PJGRAD( KE )
C
      END IF
C
  130 CONTINUE
      NE1 = NE2 + 1
      NE2 = NE2 + NOFVEE( INE + 1 )
   90 CONTINUE
C
      DO 140 IH = 1 , 2
      DO 140 IS = 1 , NS
      RGRAD( IS , IH ) = RGRAD( IS , IH ) * SAREA( IS )
      UGRAD( IS , IH ) = UGRAD( IS , IH ) * SAREA( IS )
      VGRAD( IS , IH ) = VGRAD( IS , IH ) * SAREA( IS )
      PGRAD( IS , IH ) = PGRAD( IS , IH ) * SAREA( IS )
  140 CONTINUE
C
C=====================================================================
C
C --- CALL THE MONOTONICITY LIMITER ----------------------------------
C
      CALL MONOTN
C
C=====================================================================
C
C
C --- EXIT POINT FROM SUBROUTINE -------------------------------------
C
C     ------
      RETURN
C     ------
C
C     ---
      END
```

Thu Jul 1 14:15:55 1993 gradhd.f SUBROUTINE LUDCMP

```
      SUBROUTINE LUDCMP(A,N,NP,INDX,D)
C
C-----------------------------------------------------------------I
C                                                                 I
C      PERFORM AN L U DECOMPOSITION OF THE A MATRIX               I
C                                                                 I
C-----------------------------------------------------------------I
C
      PARAMETER (NMAX=100,TINY=1.0E-20)
      DIMENSION A(NP,NP),INDX(N),VV(NMAX)
      D=1.
      DO 12 I=1,N
        AAMAX=0.
        DO 11 J=1,N
          IF (ABS(A(I,J)).GT.AAMAX) AAMAX=ABS(A(I,J))
11      CONTINUE
        IF (AAMAX.EQ.0.) PAUSE 'Singular matrix.'
        VV(I)=1./AAMAX
12    CONTINUE
      DO 19 J=1,N
        IF (J.GT.1) THEN
          DO 14 I=1,J-1
            SUM=A(I,J)
            IF (I.GT.1)THEN
              DO 13 K=1,I-1
                SUM=SUM-A(I,K)*A(K,J)
13            CONTINUE
              A(I,J)=SUM
            ENDIF
14        CONTINUE
        ENDIF
        AAMAX=0.
        DO 16 I=J,N
          SUM=A(I,J)
          IF (J.GT.1)THEN
            DO 15 K=1,J-1
              SUM=SUM-A(I,K)*A(K,J)
```

```
15            CONTINUE
              A(I,J)=SUM
            ENDIF
            DUM=VV(I)*ABS(SUM)
            IF (DUM.GE.AAMAX) THEN
              IMAX=I
              AAMAX=DUM
            ENDIF
16        CONTINUE
          IF (J.NE.IMAX)THEN
            DO 17 K=1,N
              DUM=A(IMAX,K)
              A(IMAX,K)=A(J,K)
              A(J,K)=DUM
17          CONTINUE
            D=-D
            VV(IMAX)=VV(J)
          ENDIF
          INDX(J)=IMAX
          IF(J.NE.N)THEN
            IF(A(J,J).EQ.0.)A(J,J)=TINY
            DUM=1./A(J,J)
            DO 18 I=J+1,N
              A(I,J)=A(I,J)*DUM
18          CONTINUE
          ENDIF
19      CONTINUE
        IF(A(N,N).EQ.0.)A(N,N)=TINY
        RETURN
        END
c
```

Thu Jul 1 14:15:55 1993 gradhd.f SUBROUTINE LUBKSB

```
        SUBROUTINE LUBKSB(A,N,NP,INDX,B)
        DIMENSION A(NP,NP),INDX(N),B(N)
        II=0
        DO 12 I=1,N
          LL=INDX(I)
          SUM=B(LL)
          B(LL)=B(I)
          IF (II.NE.0)THEN
            DO 11 J=II,I-1
              SUM=SUM-A(I,J)*B(J)
11          CONTINUE
          ELSE IF (SUM.NE.0.) THEN
            II=I
          ENDIF
          B(I)=SUM
12      CONTINUE
        DO 14 I=N,1,-1
          SUM=B(I)
          IF(I.LT.N)THEN
            DO 13 J=I+1,N
              SUM=SUM-A(I,J)*B(J)
13          CONTINUE
          ENDIF
          B(I)=SUM/A(I,I)
14      CONTINUE
        RETURN
        END
```

```
      SUBROUTINE FIRST
C
C---------------------------------------------------------------I
C                                                               I
C      FIRST IS USED TO FIND THE LEFT AND RIGHT INTERFACE       I
C            QUANTITIES TO FIRST ORDER WITHOUT USING EITHER THE I
C            GRADIENT OR THE CHARACTERISTICS.                   I
C                                                               I
C---------------------------------------------------------------I
C
      include     'cmsh00.h'
      include     'chyd00.h'
      include     'cint00.h'
      include     'cphs10.h'
      include     'cphs20.h'
C
C=====================================================================
C
      DO 110 IE = 1 , NE
       ISL = JE( 3 , IE )
       ISR = JE( 4 , IE )
        IJE5 = JE( 5 , IE )
           RL( IE ) =   HYDV( ISL , 1 )
           UL( IE ) =   HYDV( ISL , 2 ) * XN( IE )
     .                + HYDV( ISL , 3 ) * YN( IE )
           VL( IE ) = - HYDV( ISL , 2 ) * YN( IE )
     .                + HYDV( ISL , 3 ) * XN( IE )
           PL( IE ) =   HYDV( ISL , 4 )
C
C --- EDGES IN THE COMPUTATIONAL DOMAIN ------------------------------
C
      IF( IJE5 . EQ . 0 ) THEN
           RR( IE ) =   HYDV( ISR , 1 )
           UR( IE ) =   HYDV( ISR , 2 ) * XN( IE )
     .                + HYDV( ISR , 3 ) * YN( IE )
           VR( IE ) = - HYDV( ISR , 2 ) * YN( IE )
     .                + HYDV( ISR , 3 ) * XN( IE )
           PR( IE ) =   HYDV( ISR , 4 )
C
C --- EDGES ON THE BOUNDARY WITH ENFORCED CONDITIONS ------------------
C
C          IJE5 = 6 A WALL WITH REFLECTING NORMAL COMPONENTS
C               = 7 SUPERSONIC OUTFLOW ZERO NORMAL DERIVATIVE
C               = 8 INFLOW WITH PRESPECIFIED VALUES (RIN,UIN,VIN,PIN)
C
      ELSEIF( IJE5 . EQ . 8 ) THEN
           RR( IE ) =   RIN
           UR( IE ) =   UIN * XN( IE ) + VIN * YN( IE )
           VR( IE ) = - UIN * YN( IE ) + VIN * XN( IE )
           PR( IE ) =   PIN
C
      ELSEIF( IJE5 . EQ . 7 ) THEN
           RR( IE ) =   RL( IE )
           UR( IE ) =   UL( IE )
           VR( IE ) =   VL( IE )
           PR( IE ) =   PL( IE )
C
      ELSEIF( IJE5 . EQ . 6 . OR . IJE5 . EQ . 5 ) THEN
           RR( IE ) =   RL( IE )
           UR( IE ) = - UL( IE )
           VR( IE ) =   VL( IE )
           PR( IE ) =   PL( IE )
C
      END IF
  110 CONTINUE
C
C=====================================================================
C
C --- EXIT POINT FROM SUBROUTINE -------------------------------------
C
C     ------
      RETURN
C     ------
C
```

```
C     ---
      END
```

Thu Jul 1 14:15:55 1993 gradhd.f SUBROUTINE FCHART

```
      SUBROUTINE FCHART
C
C----------------------------------------------------------------I
C                                                                I
C       FCHART LIMITS THE PROJECTED INTERFACE VALUES ACCORDING TO I
C       CHARACTERISTICS.                                         I
C                                                                I
C----------------------------------------------------------------I
C
      include      'cmsh00.h'
      include      'chyd00.h'
      include      'cint00.h'
      include      'cphs10.h'
      include      'cphs20.h'
C
C======================================================================
C
      REAL ZZLEFT(MBP),ZOLEFT(MBP),ZMLEFT(MBP)
      REAL ZZRIGT(MBP),ZORIGT(MBP),ZPRIGT(MBP)
      REAL UPLEFT(MBP),UMLEFT(MBP),URLEFT(MBP),SQGMTL(MBP)
      REAL UPRIGT(MBP),UMRIGT(MBP),URRIGT(MBP),SQGMTR(MBP)
      REAL UVLEFT(MBP),UVRIGT(MBP),CNLEFT(MBP),CNRIGT(MBP)
      REAL RLEFTT(MBP),ULEFTT(MBP),VLEFTT(MBP),PLEFTT(MBP)
      REAL RRIGHT(MBP),URIGHT(MBP),VRIGHT(MBP),PRIGHT(MBP)
C
C======================================================================
C
      NE1 = 1
      NE2 = NOFVEE( 1 )
      DO 90 INE = 1 , NVEEE
C
      DO 110 IE = NE1 , NE2
          KE = IE - NE1 + 1
C
       ISL = JE( 3 , IE )
       ISR = JE( 4 , IE )
       GAMAL( KE ) = HYDV( ISL , 5 )
C
       CNLFTS = GAMAL( KE ) * HYDV( ISL , 4 ) / HYDV( ISL , 1 )
       CNLFT = SQRT( CNLFTS )
       UVLFT = HYDV( ISL , 2 ) * XXN( IE ) +
     .         HYDV( ISL , 3 ) * YYN( IE )
C
       IJE5 = JE( 5 , IE )
       IF( IJE5 . EQ . 0 ) THEN
C
       GAMAR( KE ) = HYDV( ISR , 5 )
       CNRGTS = GAMAR( KE ) * HYDV( ISR , 4 ) / HYDV( ISR , 1 )
       CNRGT = SQRT( CNRGTS )
C
       UVRGT = HYDV( ISR , 2 ) * XXN( IE ) +
     .         HYDV( ISR , 3 ) * YYN( IE )
C
       ELSE
C
        GAMAR( KE ) = GAMAL( KE )
        CNRGT = CNLFT
        UVRGT = UVLFT
C
       END IF
C
         CNLEFT( KE ) = CNLFT
         CNRIGT( KE ) = CNRGT
C
         UVLEFT( KE ) = UVLFT
         UVRIGT( KE ) = UVRGT
C
  110 CONTINUE
C
```

```
      DO 130 KE = 1 , NOFVEE( INE )
C
      ZZLEFT( KE ) =   .5 * ( UVLEFT( KE ) + CNLEFT( KE ) ) * DTT
      ZZRIGT( KE ) = - .5 * ( UVRIGT( KE ) - CNRIGT( KE ) ) * DTT
C
 130  CONTINUE
C
C  CHARACTERISTICS LOCATIONS
C
      DO 140 KE = 1 , NOFVEE( INE )
C
      IF( ZZLEFT( KE ) . LT . 0. ) ZZLEFT( KE ) = 0.
      IF( ZZRIGT( KE ) . LT . 0. ) ZZRIGT( KE ) = 0.
C
 140  CONTINUE
C
C      DO 150 KE = 1 , NOFVEE( INE )
C
C      ZOLEFT( KE ) =   .5 * UVLEFT( KE ) * DTT
C      ZORIGT( KE ) = - .5 * UVRIGT( KE ) * DTT
C      ZPRIGT( KE ) = - .5 * ( UVRIGT( KE ) + CNRIGT( KE ) ) * DTT
C      ZMLEFT( KE ) =   .5 * ( UVLEFT( KE ) - CNLEFT( KE ) ) * DTT
C
C 150  CONTINUE
C
C  FIRST GUESS LEFT AND RIGHT VARIABLES, LINEAR INTERPOLATON
C
      DO 160 IE = NE1 , NE2
         KE = IE - NE1 + 1
C
       ISL = JE( 3 , IE )
       ISR = JE( 4 , IE )
C
       XX = XMIDL( IE ) - ZZLEFT( KE ) * XXN( IE ) - XS( 1 , ISL )
       YY = YMIDL( IE ) - ZZLEFT( KE ) * YYN( IE ) - XS( 2 , ISL )
C
       HRRL =  HYDV( ISL , 1 ) +
     .         RGRAD( ISL , 1 ) * XX + RGRAD( ISL , 2 ) * YY
       HUUL =  HYDV( ISL , 2 ) +
     .         UGRAD( ISL , 1 ) * XX + UGRAD( ISL , 2 ) * YY
       HVVL =  HYDV( ISL , 3 ) +
     .         VGRAD( ISL , 1 ) * XX + VGRAD( ISL , 2 ) * YY
       HPPL =  HYDV( ISL , 4 ) +
     .         PGRAD( ISL , 1 ) * XX + PGRAD( ISL , 2 ) * YY
C
       GMTLFT = GAMAL( KE ) * HRRL * HPPL
       SQGMTL( KE ) = SQRT( GMTLFT )
C
C       UMLFT = 0.
C       IF( UVLEFT( KE ) - CNLEFT( KE ) . GT . 0. ) THEN
C       XX = ( ZMLEFT( KE ) - ZZLEFT( KE ) ) * XXN( IE )
C       YY = ( ZMLEFT( KE ) - ZZLEFT( KE ) ) * YYN( IE )
C       UUU = UGRAD( ISL , 1 ) * XX + UGRAD( ISL , 2 ) * YY
C       VVV = VGRAD( ISL , 1 ) * XX + VGRAD( ISL , 2 ) * YY
C       UVU = UUU * XXN( IE ) + VVV * YYN( IE )
C       PPP = PGRAD( ISL , 1 ) * XX + PGRAD( ISL , 2 ) * YY
C       UMLFT =   .5 * ( UVU - PPP / SQGMTL( KE ) ) / SQGMTL( KE )
C       END IF
C
C       URLFT = 0.
C       IF( UVLEFT( KE ) . GT . 0. ) THEN
C       XX = ( ZOLEFT( KE ) - ZZLEFT( KE ) ) * XXN( IE )
C       YY = ( ZOLEFT( KE ) - ZZLEFT( KE ) ) * YYN( IE )
C       PPP = PGRAD( ISL , 1 ) * XX + PGRAD( ISL , 2 ) * YY
C       XX = XMIDL( IE ) - ZOLEFT( KE ) * XXN( IE ) - XS( 1 , ISL )
C       YY = YMIDL( IE ) - ZOLEFT( KE ) * YYN( IE ) - XS( 2 , ISL )
C       RRRR = HYDV( ISL , 1 ) +
C    .         RGRAD( ISL , 1 ) * XX + RGRAD( ISL , 2 ) * YY
C       URLFT = PPP / GMTLFT + 1. / HRRL - 1. / RRRR
C       END IF
C
       IJE5 = JE( 5 , IE )
       IF( IJE5 . EQ . 0 ) THEN
C
```

```
            XX = XMIDL( IE ) + ZZRIGT( KE ) * XXN( IE ) - XS( 1 , ISR )
            YY = YMIDL( IE ) + ZZRIGT( KE ) * YYN( IE ) - XS( 2 , ISR )
C
            HRRR =  HYDV( ISR , 1 ) +
     .              RGRAD( ISR , 1 ) * XX + RGRAD( ISR , 2 ) * YY
            HUUR =  HYDV( ISR , 2 ) +
     .              UGRAD( ISR , 1 ) * XX + UGRAD( ISR , 2 ) * YY
            HVVR =  HYDV( ISR , 3 ) +
     .              VGRAD( ISR , 1 ) * XX + VGRAD( ISR , 2 ) * YY
            HPPR =  HYDV( ISR , 4 ) +
     .              PGRAD( ISR , 1 ) * XX + PGRAD( ISR , 2 ) * YY
C
            GMTRGT = GAMAR( KE ) * HRRR * HPPR
            SQGMTR( KE ) = SQRT( GMTRGT )
C
C            UPRGT = 0.
C            IF( UVRIGT( KE ) + CNRIGT( KE ) . LT . 0. ) THEN
C            XX = ( ZZRIGT( KE ) - ZPRIGT( KE ) ) * XXN( IE )
C            YY = ( ZZRIGT( KE ) - ZPRIGT( KE ) ) * YYN( IE )
C            UUU = UGRAD( ISR , 1 ) * XX + UGRAD( ISR , 2 ) * YY
C            VVV = VGRAD( ISR , 1 ) * XX + VGRAD( ISR , 2 ) * YY
C            UVU = UUU * XXN( IE ) + VVV * YYN( IE )
C            PPP = PGRAD( ISR , 1 ) * XX + PGRAD( ISR , 2 ) * YY
C            UPRGT = - .5 * ( UVU + PPP / SQGMTR( KE ) ) / SQGMTR( KE )
C            END IF
C
C            URRGT = 0.
C            IF( UVRIGT( KE ) . LT . 0. ) THEN
C            XX = ( ZZRIGT( KE ) - ZORIGT( KE ) ) * XXN( IE )
C            YY = ( ZZRIGT( KE ) - ZORIGT( KE ) ) * YYN( IE )
C            PPP = PGRAD( ISR , 1 ) * XX + PGRAD( ISR , 2 ) * YY
C            XX = XMIDL( IE ) + ZORIGT( KE ) * XXN( IE ) - XS( 1 , ISR )
C            YY = YMIDL( IE ) + ZORIGT( KE ) * YYN( IE ) - XS( 2 , ISR )
C            RRRR = HYDV( ISR , 1 ) +
C    .              RGRAD( ISR , 1 ) * XX + RGRAD( ISR , 2 ) * YY
C            URRGT = PPP / GMTRGT + 1. / HRRR - 1. / RRRR
C            END IF
C
          ELSE
C
            HRRR =  HRRL
            HUUR =  HUUL
            HVVR =  HVVL
            HPPR =  HPPL
C
C            UPRGT = UMLFT
C            URRGT = URLFT
C
          END IF
C
          RRL( KE ) = HRRL
          UUL( KE ) =   HUUL * XN( IE ) + HVVL * YN( IE )
          VVL( KE ) = - HUUL * YN( IE ) + HVVL * XN( IE )
          PPL( KE ) = HPPL
C
          RRR( KE ) = HRRR
          UUR( KE ) =   HUUR * XN( IE ) + HVVR * YN( IE )
          VVR( KE ) = - HUUR * YN( IE ) + HVVR * XN( IE )
          PPR( KE ) = HPPR
C
C           UMLEFT( KE ) = UMLFT
C           URLEFT( KE ) = URLFT
C
C           UPRIGT( KE ) = UPRGT
C           URRIGT( KE ) = URRGT
C
  160   CONTINUE
C
C  FINAL VALUES FOR RIGHT AND LEFT STATES
C
C          DO 180 KE = 1 , NOFVEE( INE )
C
C            GMTLFT = SQGMTL( KE ) * SQGMTL( KE )
C            GMTRGT = SQGMTR( KE ) * SQGMTR( KE )
```

```
C
C       RRL( KE ) = 1. / ( 1. / RRL( KE ) - ( UMLEFT( KE ) +
C      .                                     URLEFT( KE ) ) )
C       UUL( KE ) = UUL( KE ) - SQGMTL( KE ) * UMLEFT( KE )
C       PPL( KE ) = PPL( KE ) + GMTLFT * UMLEFT( KE )
C
C       RRR( KE ) = 1. / ( 1. / RRR( KE ) - ( UPRIGT( KE ) +
C      .                                     URRIGT( KE ) ) )
C       UUR( KE ) = UUR( KE ) + SQGMTR( KE ) * UPRIGT( KE )
C       PPR( KE ) = PPR( KE ) + GMTRGT * UPRIGT( KE )
C
C 180  CONTINUE
C
      DO 200 IE = NE1 , NE2
         KE = IE - NE1 + 1
C
       ISL = JE( 3 , IE )
       ISR = JE( 4 , IE )
C
       IJE5 = JE( 5 , IE )
C
C --- PROJECTED VALUES ON THE LEFT SIDE OF THE INTERFACE ---------------
C
       RL( IE ) = RRL( KE )
       UL( IE ) = UUL( KE )
       VL( IE ) = VVL( KE )
       PL( IE ) = PPL( KE )
C
C --- PROJECTED VALUES ON THE RIGHT SIDE OF THE INTERFACE --------------
C
C --- EDGES IN THE COMPUTATIONAL DOMAIN --------------------------------
C
      IF( IJE5 . EQ . 0 ) THEN
         RR( IE ) = RRR( KE )
         UR( IE ) = UUR( KE )
         VR( IE ) = VVR( KE )
         PR( IE ) = PPR( KE )
C
C --- EDGES ON THE BOUNDARY -------------------------------------------
C
      ELSEIF( IJE5 . EQ . 8 ) THEN
         RR( IE ) =   RIN
         UR( IE ) =   UIN * XN( IE ) + VIN * YN( IE )
         VR( IE ) = - UIN * YN( IE ) + VIN * XN( IE )
         PR( IE ) =   PIN
C
      ELSEIF( IJE5 . EQ . 7 ) THEN
         RR( IE ) = RL( IE )
         UR( IE ) = UL( IE )
         VR( IE ) = VL( IE )
         PR( IE ) = PL( IE )
C
      ELSEIF( IJE5 . EQ . 6 . OR . IJE5 . EQ . 5 ) THEN
         RR( IE ) = RL( IE )
         UR( IE ) = - UL( IE )
         VR( IE ) = VL( IE )
         PR( IE ) = PL( IE )
C
      END IF
  200 CONTINUE
C
      NE1 = NE2 + 1
      NE2 = NE2 + NOFVEE( INE + 1 )
   90 CONTINUE
C
C=====================================================================
C
C --- EXIT POINT FROM SUBROUTINE ---------------------------------------
C
C     ------
      RETURN
C     ------
C
C     ---
```

```
      END
```

Thu Jul 1 14:15:55 1993 gradhd.f SUBROUTINE PRLCTN

```
      SUBROUTINE PRLCTN
C
C--------------------------------------------------------------I
C                                                              I
C       PRLCTN INITIALIZE PARTICLES LOCATION IN THE COMPUTATION I
C              DOMAIN FOR THE FIRST TIME                       I
C                                                              I
C--------------------------------------------------------------I
C
      include     'cmsh00.h'
      include     'chyd00.h'
      include     'cint00.h'
      include     'cphs10.h'
      include     'cphs20.h'
C
C=====================================================================
C
      IPT = 0
      DO 110 IPRTCL = 1 , NPT
C
      IDUM = 0
      DO 130 IS = 1 , NS
      IF( IDUM . EQ . 0 ) THEN
C
      IV1 = JS( 1 , IS )
      IV2 = JS( 2 , IS )
      IV3 = JS( 3 , IS )
C
      X1 = XV( 1 , IV1 )
      Y1 = XV( 2 , IV1 )
      X2 = XV( 1 , IV2 )
      Y2 = XV( 2 , IV2 )
C
      XX = ( X2 - X1 )
      XXP = ( XPRTCL( 1 , IPRTCL ) - X1 )
C
      YY = ( Y2 - Y1 )
      YYP = ( XPRTCL( 2 , IPRTCL ) - Y1 )
C
      A1 = XX * YYP - YY * XXP
C
      X1 = XV( 1 , IV2 )
      Y1 = XV( 2 , IV2 )
      X2 = XV( 1 , IV3 )
      Y2 = XV( 2 , IV3 )
C
      XX = ( X2 - X1 )
      XXP = ( XPRTCL( 1 , IPRTCL ) - X1 )
C
      YY = ( Y2 - Y1 )
      YYP = ( XPRTCL( 2 , IPRTCL ) - Y1 )
C
      A2 = XX * YYP - YY * XXP
C
      X1 = XV( 1 , IV3 )
      Y1 = XV( 2 , IV3 )
      X2 = XV( 1 , IV1 )
      Y2 = XV( 2 , IV1 )
C
      XX = ( X2 - X1 )
      XXP = ( XPRTCL( 1 , IPRTCL ) - X1 )
C
      YY = ( Y2 - Y1 )
      YYP = ( XPRTCL( 2 , IPRTCL ) - Y1 )
C
      A3 = XX * YYP - YY * XXP
C
      IA1 = INT( SIGN( 1.1 , A1 ) )
      IA2 = INT( SIGN( 1.1 , A2 ) )
      IA3 = INT( SIGN( 1.1 , A3 ) )
```

```
            IAJ = IA1 + IA2 + IA3
      C
            IF( IAJ . EQ . 3 )  THEN
            IPT = IPT + 1
            IJKPRT( IPT ) = IS
            XPRTCL( 1 , IPT ) = XPRTCL( 1 , IPRTCL )
            XPRTCL( 2 , IPT ) = XPRTCL( 2 , IPRTCL )
      C      PRINT *, XPRTCL(1,IPT),XPRTCL(2,IPT),IJKPRT(IPT)
            IDUM = 1
            END IF
            END IF
      C
        130 CONTINUE
        110 CONTINUE
            NPT = IPT
      C       PRINT *,   NPT,(XPRTCL(1,IPT),XPRTCL(2,IPT),IPT=1,NPT)
      C       WRITE (10,*)  NPT,(XPRTCL(1,IPT),XPRTCL(2,IPT),IPT=1,NPT)
      C
      C
      C --- EXIT POINT FROM SUBROUTINE ------------------------------------
      C
      C     ------
            RETURN
      C     ------
      C
      C     ---
            END
```

Thu Jul 1 14:15:55 1993 gradhd.f SUBROUTINE PRPTHC

```
            SUBROUTINE PRPTHC
      C
      C--------------------------------------------------------------I
      C                                                              I
      C      PRPTHC TRACE PARTICLES PATH IN THE COMPUTATION DOMAIN   I
      C                                                              I
      C--------------------------------------------------------------I
      C
            include     'cmsh00.h'
            include     'chyd00.h'
            include     'cint00.h'
            include     'cphs10.h'
            include     'cphs20.h'
      C
            DO 110 IPRTCL = 1 , NPT
            KFIND = 0
      C
            DO 110 IK = 1 , 3
            KFIND = 0
            IJES = 0
            IS = IJKPRT( IPRTCL )
            XP = XPRTCL( 1 , IPRTCL )
            YP = XPRTCL( 2 , IPRTCL )
      C
            DO 120 IJ = 1 , 3
            IE = JS( IJ + 3 , IS )
      C
            IF( IE . GT . 0 ) THEN
      C
            IV1 = JE( 1 , IE )
            IV2 = JE( 2 , IE )
      C
            X1 = XV( 1 , IV1 )
            Y1 = XV( 2 , IV1 )
            X2 = XV( 1 , IV2 )
            Y2 = XV( 2 , IV2 )
      C
            XX = ( X2 - X1 )
            XXP = ( XP - X1 )
      C
            YY = ( Y2 - Y1 )
            YYP = ( YP - Y1 )
      C
            A = XX * YYP - YY * XXP
```

```
      IF( A . LT . 0. ) THEN
      IJKPRT( IPRTCL ) = JE( 4 , IE )
      IJE5 = JE( 5 , IE )
      XREV = 1. / XE( 1 , IE )
      KFIND = KFIND + 1
      END IF
C
      ELSE
C
      IV1 = JE( 2 , - IE )
      IV2 = JE( 1 , - IE )
C
      X1 = XV( 1 , IV1 )
      Y1 = XV( 2 , IV1 )
      X2 = XV( 1 , IV2 )
      Y2 = XV( 2 , IV2 )
C
      XX = ( X2 - X1 )
      XXP = ( XP - X1 )
C
      YY = ( Y2 - Y1 )
      YYP = ( YP - Y1 )
C
      A = XX * YYP - YY * XXP
      IF( A . LT . 0. ) THEN
      IJKPRT( IPRTCL ) = JE( 3 , - IE )
      IJE5 = JE( 5 , - IE )
      XREV = 1. / XE( 1 , - IE )
      KFIND = KFIND + 1
      END IF
      END IF
C
  120 CONTINUE
C
      IF( KFIND . GT . 0 . AND . IJE5 . NE . 0 ) THEN
      IJKPRT( IPRTCL ) = IS
C
      AA = X2 - X1
      BB = Y2 - Y1
      CC = XP - X1
      DD = YP - Y1
      TREV = ( CC * AA + DD * BB ) * XREV * XREV
      IF( BB . NE . 0. ) THEN
      XPRTCP = X1 + TREV * AA
      XPRTCL( 1 , IPRTCL ) = XP + 1.1 * ( XPRTCP - XP )
      END IF
      IF( AA . NE . 0. ) THEN
      YPRTCP = Y1 + TREV * BB
      XPRTCL( 2 , IPRTCL ) = YP + 1.1 * ( YPRTCP - YP )
      END IF
C
      END IF
  110 CONTINUE
C
      DO 180 IPRTCL = 1 , NPT
C
      IS = IJKPRT( IPRTCL )
      UPRTCL = HYDV( IS , 2 )
      VPRTCL = HYDV( IS , 3 )
C
      XPRTCL( 1 , IPRTCL ) = XPRTCL( 1 , IPRTCL ) + UPRTCL * DTT
      XPRTCL( 2 , IPRTCL ) = XPRTCL( 2 , IPRTCL ) + VPRTCL * DTT
      WPRTCL( 1 , IPRTCL ) = UPRTCL
      WPRTCL( 2 , IPRTCL ) = VPRTCL
C
      DO 180 IK = 1 , 3
      IS = IJKPRT( IPRTCL )
      XP = XPRTCL( 1 , IPRTCL )
      YP = XPRTCL( 2 , IPRTCL )
      KFIND = 0
      IJE5 = 0
C
      DO 170 IJ = 1 , 3
      IE = JS( IJ + 3 , IS )
```

```
C
      IF( IE . GT . 0 ) THEN
C
      IV1 = JE( 1 , IE )
      IV2 = JE( 2 , IE )
C
      X1 = XV( 1 , IV1 )
      Y1 = XV( 2 , IV1 )
      X2 = XV( 1 , IV2 )
      Y2 = XV( 2 , IV2 )
C
      XX = ( X2 - X1 )
      XXP = ( XP - X1 )
C
      YY = ( Y2 - Y1 )
      YYP = ( YP - Y1 )
C
      A = XX * YYP - YY * XXP
      IF( A . LT . 0. )  THEN
      IJKPRT( IPRTCL ) = JE( 4 , IE )
      IJE5 = JE( 5 , IE )
      XREV = 1. / XE( 1 , IE )
      KFIND = KFIND + 1
      END IF
C
      ELSE
C
      IV1 = JE( 2 , - IE )
      IV2 = JE( 1 , - IE )
C
      X1 = XV( 1 , IV1 )
      Y1 = XV( 2 , IV1 )
      X2 = XV( 1 , IV2 )
      Y2 = XV( 2 , IV2 )
C
      XX = ( X2 - X1 )
      XXP = ( XP - X1 )
C
      YY = ( Y2 - Y1 )
      YYP = ( YP - Y1 )
C
      A = XX * YYP - YY * XXP
      IF( A . LT . 0. )  THEN
      IJKPRT( IPRTCL ) = JE( 3 , - IE )
      IJE5 = JE( 5 , - IE )
      XREV = 1. / XE( 1 , - IE )
      KFIND = KFIND + 1
      END IF
C
      END IF
  170 CONTINUE
C
      IF( KFIND . GT . 0 . AND . IJE5 . NE . 0 ) THEN
      IJKPRT( IPRTCL ) = IS
C
      AA = X2 - X1
      BB = Y2 - Y1
      CC = XP - X1
      DD = YP - Y1
      TREV = ( CC * AA + DD * BB ) * XREV * XREV
      IF( BB . NE . 0. ) THEN
      XPRTCP = X1 + TREV * AA
      XPRTCL( 1 , IPRTCL ) = XP + 1.1 * ( XPRTCP - XP )
      END IF
      IF( AA . NE . 0. ) THEN
      YPRTCP = Y1 + TREV * BB
      XPRTCL( 2 , IPRTCL ) = YP + 1.1 * ( YPRTCP - YP )
      END IF
C
      END IF
  180 CONTINUE
C
C
C --- EXIT POINT FROM SUBROUTINE ---------------------------------------
```

```
C
C     ------
      RETURN
C     ------
C
C     ---
      END
```

Thu Jul 1 14:16:08 1993 adaphd.f Module List - alphabetical order

```
      SUBROUTINE VERCEN( IT )
C
C---------------------------------------------------------------I
C                                                               I
C          VERCEN ADD A VERTEX IN THE IT TRIANGLE. THE VERTEX   I
C          IS ADDED IN THE CENTROID OF THE TRIANGLE.            I
C                                                               I
C---------------------------------------------------------------I
C
          IMPLICIT REAL (A-H,O-Z)
C
      include     'cmsh00.h'
      include     'chyd00.h'
      include     'cint00.h'
      include     'cphs10.h'
      include     'cphs20.h'
C
C         SET UP THE NEW TRIANGLE BOOKKEEPING.
C
          IV1 = JS( 1 , IT )
          IV2 = JS( 2 , IT )
          IV3 = JS( 3 , IT )
C
          IE1 = JS( 4 , IT )
          IE2 = JS( 5 , IT )
          IE3 = JS( 6 , IT )
          IE1A = IABS( IE1 )
          IE2A = IABS( IE2 )
          IE3A = IABS( IE3 )
C
C       PUT IN NEW TRIANGLES
C
          NV = NV + 1
          XV( 1 , NV ) = ( XV( 1 , IV1 ) + XV( 1 , IV2 ) +
     .                                     XV( 1 , IV3 ) ) * THIRD
          XV( 2 , NV ) = ( XV( 2 , IV1 ) + XV( 2 , IV2 ) +
     .                                     XV( 2 , IV3 ) ) * THIRD
          JV( 1 , NV ) = 0
C
          DO 110 IR = 1 , MHQ
          HYDVVV( NV , IR ) = ( HYDVVV( IV1 , IR ) +
     .                          HYDVVV( IV2 , IR ) +
     .                          HYDVVV( IV3 , IR ) ) * THIRD
 110      CONTINUE
C
          NE = NE + 1
          JE( 1 , NE ) = NV
          JE( 2 , NE ) = IV1
          JE( 5 , NE ) = 0
          NE = NE + 1
          JE( 1 , NE ) = NV
          JE( 2 , NE ) = IV2
          JE( 5 , NE ) = 0
          NE = NE + 1
          JE( 1 , NE ) = NV
          JE( 2 , NE ) = IV3
          JE( 5 , NE ) = 0
          NEM1 = NE - 1
          NEM2 = NE - 2
C
C         TRIANGLE ONE, THE ORIGINAL IT.
C
          JS( 3 , IT ) = NV
          JS( 5 , IT ) = - NEM1
          JS( 6 , IT ) = NEM2
C
C         TRIANGLE TWO.
C
          NS = NS + 1
          JS( 1 , NS ) = IV2
          JS( 2 , NS ) = IV3
          JS( 3 , NS ) = NV
          JS( 4 , NS ) = IE2
```

```
      JS( 5 , NS ) = - NE
      JS( 6 , NS ) = NEM1
C
C     TRIANGLE THREE.
C
      NS = NS + 1
      JS( 1 , NS ) = IV3
      JS( 2 , NS ) = IV1
      JS( 3 , NS ) = NV
      JS( 4 , NS ) = IE3
      JS( 5 , NS ) = - NEM2
      JS( 6 , NS ) = NE
C
C     NOW FIX THE LEFT AND RIGHT FOR EDGES.
C
      NSM1 = NS - 1
      IF( JE( 4 , IE2A ) . EQ . IT ) JE( 4 , IE2A ) = NSM1
      IF( JE( 3 , IE2A ) . EQ . IT ) JE( 3 , IE2A ) = NSM1
      IF( JE( 4 , IE3A ) . EQ . IT ) JE( 4 , IE3A ) = NS
      IF( JE( 3 , IE3A ) . EQ . IT ) JE( 3 , IE3A ) = NS
      JE( 4 , NEM2 ) = NS
      JE( 3 , NEM2 ) = IT
      JE( 4 , NEM1 ) = IT
      JE( 3 , NEM1 ) = NSM1
      JE( 4 , NE ) = NSM1
      JE( 3 , NE ) = NS
C
      JV( 2 , NV ) = NE
C
      XSAREA = XS( 3 , IT ) * THIRD
      XS( 3 , IT ) = XSAREA
      XS( 3 , NSM1 ) = XSAREA
      XS( 3 , NS ) = XSAREA
C
      XS( 1 , IT ) = ( XV( 1 , IV1 ) + XV( 1 , IV2 ) +
     .                 XV( 1 , NV ) ) * THIRD
      XS( 1 , NSM1 ) = ( XV( 1 , IV2 ) + XV( 1 , IV3 ) +
     .                   XV( 1 , NV ) ) * THIRD
      XS( 1 , NS ) = ( XV( 1 , IV3 ) + XV( 1 , IV1 ) +
     .                 XV( 1 , NV ) ) * THIRD
      XS( 2 , IT ) = ( XV( 2 , IV1 ) + XV( 2 , IV2 ) +
     .                 XV( 2 , NV ) ) * THIRD
      XS( 2 , NSM1 ) = ( XV( 2 , IV2 ) + XV( 2 , IV3 ) +
     .                   XV( 2 , NV ) ) * THIRD
      XS( 2 , NS ) = ( XV( 2 , IV3 ) + XV( 2 , IV1 ) +
     .                 XV( 2 , NV ) ) * THIRD
C
      XSAREA = 1. / XS( 3 , IT )
      SAREA( IT ) = XSAREA
      SAREA( NS ) = XSAREA
      SAREA( NSM1 ) = XSAREA
C
      DO 630 IR = 1 , MHQ
      HYDV( IT , IR ) = ( HYDVVV( IV1 , IR ) +
     .                    HYDVVV( IV2 , IR ) +
     .                    HYDVVV( NV , IR ) ) * THIRD
      HYDV( NS , IR ) = ( HYDVVV( IV3 , IR ) +
     .                    HYDVVV( IV1 , IR ) +
     .                    HYDVVV( NV , IR ) ) * THIRD
      HYDV( NSM1 , IR ) = ( HYDVVV( IV2 , IR ) +
     .                      HYDVVV( IV3 , IR ) +
     .                      HYDVVV( NV , IR ) ) * THIRD
  630 CONTINUE
C
      HDUM            = 1. / ( HYDV( IT , 1 ) + 1.E-12 )
      HYDV( IT , 2 ) = HYDV( IT , 2 ) * HDUM
      HYDV( IT , 3 ) = HYDV( IT , 3 ) * HDUM
      HYDV( IT , 4 ) = ( HYDV( IT , 4 )  -
     .                .5 * HYDV( IT , 1 ) *
     .  ( HYDV( IT , 2 ) * HYDV( IT , 2 ) +
     .    HYDV( IT , 3 ) * HYDV( IT , 3 ) ) ) *
     .                ( HYDV( IT , 5 ) - 1. )
C
      HDUM            = 1. / ( HYDV( NS , 1 ) + 1.E-12 )
```

```
          HYDV( NS , 2 ) = HYDV( NS , 2 ) * HDUM
          HYDV( NS , 3 ) = HYDV( NS , 3 ) * HDUM
          HYDV( NS , 4 ) = ( HYDV( NS , 4 )  -
     .                   .5 * HYDV( NS , 1 ) *
     .    ( HYDV( NS , 2 ) * HYDV( NS , 2 ) +
     .      HYDV( NS , 3 ) * HYDV( NS , 3 ) ) ) *
     .                         ( HYDV( NS , 5 ) - 1. )
C
          HDUM           = 1. / ( HYDV( NSM1 , 1 ) + 1.E-12 )
          HYDV( NSM1 , 2 ) = HYDV( NSM1 , 2 ) * HDUM
          HYDV( NSM1 , 3 ) = HYDV( NSM1 , 3 ) * HDUM
          HYDV( NSM1 , 4 ) = ( HYDV( NSM1 , 4 )  -
     .                     .5 * HYDV( NSM1 , 1 ) *
     .    ( HYDV( NSM1 , 2 ) * HYDV( NSM1 , 2 ) +
     .      HYDV( NSM1 , 3 ) * HYDV( NSM1 , 3 ) ) ) *
     .                           ( HYDV( NSM1 , 5 ) - 1. )
C
          DO 114 IR = 1 , 2
          RGRAD( NS , IR ) = RGRAD( IT , IR )
          RGRAD( NSM1 , IR ) = RGRAD( IT , IR )
C
          UGRAD( NS , IR ) = UGRAD( IT , IR )
          UGRAD( NSM1 , IR ) = UGRAD( IT , IR )
C
          VGRAD( NS , IR ) = VGRAD( IT , IR )
          VGRAD( NSM1 , IR ) = VGRAD( IT , IR )
C
          PGRAD( NS , IR ) = PGRAD( IT , IR )
          PGRAD( NSM1 , IR ) = PGRAD( IT , IR )
  114     CONTINUE
C
          JEN( 1 ) = IE1A
          JEN( 2 ) = IE2A
          JEN( 3 ) = IE3A
          JEN( 4 ) = NEM2
          JEN( 5 ) = NEM1
          JEN( 6 ) = NE
C
          DO 30 IENN = 1 , 6
          IEN = JEN( IENN )
          JV1 = JE( 1 , IEN )
          JV2 = JE( 2 , IEN )
          AX = XV( 1 , JV2 ) - XV( 1 , JV1 )
          AY = XV( 2 , JV2 ) - XV( 2 , JV1 )
          XE( 1 , IEN ) = SQRT( AX * AX + AY * AY )
          XEREV = 1. / XE( 1 , IEN )
          XN( IEN ) = AY * XEREV
          YN( IEN ) = - AX * XEREV
          ISSR = JE( 4 , IEN )
          ISSL = JE( 3 , IEN )
C
        IJE5 = JE( 5 , IEN )
        IF( IJE5 . NE . 0 ) THEN
C
          AA = XV( 1 , JV2 ) - XV( 1 , JV1 )
          BB = XV( 2 , JV2 ) - XV( 2 , JV1 )
          XEL = XS( 1 , ISSL )
          YEL = XS( 2 . ISSL )
          CC = XEL - XV( 1 , JV1 )
          DD = YEL - XV( 2 , JV1 )
          EE = ( AA * CC + BB * DD ) * XEREV * XEREV
          XER = XV( 1 , JV1 ) + AA * EE
          YER = XV( 2 , JV1 ) + BB * EE
          AX = XER - XEL
          AY = YER - YEL
          XE( 2 , IEN ) = SQRT( AX * AX + AY * AY )
          XEREV = 1. / XE( 2 , IEN )
          XXN( IEN ) = AX * XEREV
          YYN( IEN ) = AY * XEREV
          XE( 2 , IEN ) = 2. * XE( 2 , IEN )
          XYMIDL( IEN ) = .5
          XMIDL( IEN ) = XER
          YMIDL( IEN ) = YER
C
```

```
            ELSE
C
            XER = XS( 1 , ISSR )
            YER = XS( 2 , ISSR )
            XEL = XS( 1 , ISSL )
            YEL = XS( 2 , ISSL )
C
            AA = XV( 1 , JV2 ) - XV( 1 , JV1 )
            BB = XV( 2 , JV2 ) - XV( 2 , JV1 )
            CC = XEL - XER
            DD = YEL - YER
            ACA = XER - XV( 1 , JV1 )
            DBD = YER - XV( 2 , JV1 )
            EE = ( ACA * DD - DBD * CC ) / ( AA * DD - BB * CC )
            XMIDL( IEN ) = XV( 1 , JV1 ) + AA * EE
            YMIDL( IEN ) = XV( 2 , JV1 ) + BB * EE
C
            XEMID = XMIDL( IEN ) - XEL
            YEMID = YMIDL( IEN ) - YEL
C
            AX = XER - XEL
            AY = YER - YEL
            XE( 2 , IEN ) = SQRT( AX * AX + AY * AY )
            XEREV = 1. / XE( 2 , IEN )
            XXN( IEN ) = AX * XEREV
            YYN( IEN ) = AY * XEREV
C
            XYMIDL( IEN ) = SQRT( XEMID * XEMID + YEMID * YEMID ) * XEREV
C
            END IF
C
 30       CONTINUE
C
C --- EXIT POINT FROM SUBROUTINE --------------------   --------------
C
C        -----
         RETURN
C        ------
C
C        ---
         END
```

Thu Jul 1 14:16:08 1993 adaphd.f SUBROUTINE DISECT

```
         SUBROUTINE DISECT ( N , IDONE , IDUMP )
C
C-----------------------------------------------------------------I
C                                                                 I
C           DISECT DISECTS THE LINE N TO CREATE TWO NEW TRIANGLES AND I
C           A NEW VERTEX.  IF THE LINE N IS ON A SOLID BOUNDARY, ONLY I
C           ONE NEW TRIANGLE IS CREATED.                          I
C                                                                 I
C           DISECT CANNOT BE USED FOR PERIODIC SIDES.  HOWEVER, A I
C           CALL TO TSHIFT CAN BE USED TO MAKE THOSE SIDES INTERNAL I
C           BEFORE CALLING DISECT.                                I
C                                                                 I
C           INPUT:  N  - THE SIDE TO BE DISECTED.                 I
C                                                                 I
C           OUTPUT: N3 - THE SECOND HALF OF WHAT WAS N.  WHEN N IS I
C                        DISECTED, THE INDEX N IS RETAINED FOR ONE OF I
C                        THE NEW SIDES, THE OTHER IS N3.          I
C                                                                 I
C                   I1 - THE STARTING VERTEX OF THE INPUT LINE N; I
C                   I2 - THE ENDING VERTEX OF THE INPUT LINE N;   I
C                   I3 - THE THIRD VERTEX IN THE TRIANGLE TO THE RIGHT; I
C                   I4 - THE THIRD VERTEX IN THE TRIANGLE TO THE LEFT; I
C                   I5 - THE NEW VERTEX.                          I
C                                                                 I
C-----------------------------------------------------------------I
C
            IMPLICIT REAL (A-H,O-Z)
C
         include     'cmsh00.h'
         include     'chyd00.h'
```

```
      include     'cint00.h'
      include     'cphs10.h'
      include     'cphs20.h'
C
          INTEGER IS(2), IVS(2)
C
          ITRING = 0
          IDONE = 0
          IJE5 = JE( 5 , N )
          IF( IJE5 . NE . 0 ) ITRING = 1
          I5     = 0
          IEROR = 0
          EROR = 1.0E-3
C
C         FIND THE VERTICES OF THE LINE N.
C
          I1 = JE( 1 , N )
          I2 = JE( 2 , N )
          IT1    = JE( 4 , N )
          IT2    = JE( 3 , N )
C
C         FIND THE TWO VERTICES TO WHICH THE NEW LINES WILL BE DRAWN.
C         THESE ARE THE VERTICES OTHER THAN I1 AND I2 IN THE
C         TRIANGLES TO EITHER SIDE OF N.  IVS STORES THE INDEX OF
C         THESE VERTICES AND  IS STORES WHETHER THEY ARE VERTEX 1, 2
C         OR 3 IN THE TRIANGLE IT.
C
          DO 10 I = 1 , 2
             IVS(I) = 0
             IT     = JE( 5-I , N )
             IF( IT . NE . 0 ) THEN
                DO 20 J = 1 , 3
                   IV = JS( J , IT )
                   IF( IV . NE . I1 .AND. IV . NE . I2 ) THEN
                      IVS( I ) = IV
                      IS( I )  = J
                   END IF
   20           CONTINUE
             END IF
   10     CONTINUE
          I3  = IVS(1)
          I4  = IVS(2)
          IS1 = IS(1)
          IS2 = IS(2)
C
C     COMPARE OPPOSING ANGLE PAIRS IN THE QUADRILATERAL
C
          IF( ITRING . EQ . 0 ) THEN
          AX = XV( 1 , I3 ) - XV( 1 , I1 )
          AY = XV( 2 , I3 ) - XV( 2 , I1 )
          BX = XV( 1 , I4 ) - XV( 1 , I1 )
          BY = XV( 2 , I4 ) - XV( 2 , I1 )
          CX = XV( 1 , I4 ) - XV( 1 , I2 )
          CY = XV( 2 , I4 ) - XV( 2 , I2 )
          DX = XV( 1 , I3 ) - XV( 1 , I2 )
          DY = XV( 2 , I3 ) - XV( 2 , I2 )
          AI2 = AX * BY - AY * BX
          AI1 = CX * DY - CY * DX
          XLN = XE( 1 , N )
          ROUNDF = EROR * XLN * XLN
          IF( AI2 . LT . ROUNDF . OR . AI1 . LT . ROUNDF ) RETURN
          END IF
C
C         CREATE A NEW VERTEX MIDWAY ON LINE N.
C
          IDONE = 1
          NV     = NV + 1
          I5     = NV
C
C         CHANGE THE LINE N SO THAT IT STARTS AT THE SAME VERTEX,
C         BUT NOW ENDS AT I5.
C
          JE( 2 , N ) = I5
C
```

```
C       DRAW THE THREE NEW LINES, ALL ENDING AT I5.
C
        DO 30 I = 1 , 2
           IF( JE( 5 - I , N ) . NE . 0 ) THEN
        NE            = NE + 1
        JE( 1 , NE ) = IVS( I )
        JE( 2 , NE ) = I5
C
              JE( 5 , NE ) = 0
              IF( I . EQ . 1 ) N1 = NE
              IF( I . EQ . 2 ) N2 = NE
           END IF
 30     CONTINUE
C
C       WE NEED TO HANDLE THE LINE FROM I2 TO I5 SEPARATELY,
C       SINCE WE ARE NOT ADDING A LINE TO I2, BUT REPLACING
C       THE OLD ONE.
C
        NE           = NE + 1
C
        JE( 5 , NE )   = JE( 5 , N )
        N3             = NE
        JE( 3 , N3 )   = 0
        JE( 4 , N3 )   = 0
C
C       RESET THE OLD TRIANGLES AND SET UP THE NEW TRIANGLES.
C
C       N WAS ORIGINALLY DRAWN FROM I1 TO I2 -- NOW FROM I1 TO I5.
C       N1 IS THE NEW LINE FROM I3 TO I5.
C       N2 IS THE NEW LINE FROM I4 TO I5.
C       N3 IS THE NEW LINE FROM I2 TO I5.
C       NAA IS THE OLD LINE FROM I4 TO I1.
C       NBB IS THE OLD LINE FROM I1 TO I3.
C       NCC IS THE OLD LINE FROM I3 TO I2.
C       NDD IS THE OLD LINE FROM I2 TO I4.
C       THE DIRECTIONS OF LINES NAA THROUGH NDD ARE NOT
C           EXPLICITLY USED.
C
        IF(  IT1 . NE . 0 ) THEN
           NCC                    = JS( IS1 + 3 , IT1 )
           JS( IS1 + 3 , IT1 ) = N1
           J                      = MOD( IS1 , 3 ) + 1
           IF( JS( J , IT1 ) . NE . I2 ) THEN
              IEROR = 2
              J1     = J
           END IF
           JS( J , IT1 ) = I5
C
          JJ  = MOD( IS1 + 1 , 3 ) + 1
          NBB = IABS( JS( JJ + 3 , IT1 ) )
C
           NS             = NS + 1
           JS( 1 , NS ) = I2
           JS( 2 , NS ) = I5
           JS( 3 , NS ) = I3
           JS( 4 , NS ) = N3
           JS( 5 , NS ) = - N1
           JS( 6 , NS ) = NCC
           JE( 3 , N3 ) = NS
           JE( 4 , N1 ) = NS
           JE( 3 , N1 ) = IT1
           NCC            = IABS( NCC )
           IF( JE( 4 , NCC ) . EQ . IT1 ) JE( 4 , NCC ) = NS
           IF( JE( 3 , NCC ) . EQ . IT1 ) JE( 3 , NCC ) = NS
C
        END IF
C
        IF( IT2 . NE . 0 ) THEN
           J              = MOD( IS2 + 1 , 3 ) + 1
           NDD            = JS( J + 3 , IT2 )
           JS( J + 3 , IT2 ) = - N2
           IF( JS( J , IT2 ) . NE . I2 ) THEN
              IEROR = 3
              J2     = J
```

```
              END IF
              JS( J , IT2 ) = I5
C
             NAA = IABS( JS( IS2 + 3 , IT2 ) )
C
              NS           = NS + 1
              JS( 1 , NS ) = I2
              JS( 2 , NS ) = I4
              JS( 3 , NS ) = I5
              JS( 4 , NS ) = NDD
              JS( 5 , NS ) = N2
C
           IF( ITRING . EQ . 0 ) THEN
C
           JE( 1 , N3 )   = I2
           JE( 2 , N3 )   = I5
              JS( 6 , NS ) = - N3
              JE( 4 , N3 ) = NS
C
           ELSE
C
           JE( 1 , N3 )   = I5
           JE( 2 , N3 )   = I2
              JS( 6 , NS ) = N3
              JE( 3 , N3 ) = NS
C
           END IF
C
              JE( 3 , N2 ) = NS
              JE( 4 , N2 ) = IT2
              NDD          = IABS( NDD )
              IF( JE( 4 , NDD ) . EQ . IT2 ) JE( 4 , NDD ) = NS
              IF( JE( 3 , NDD ) . EQ . IT2 ) JE( 3 , NDD ) = NS
C
           END IF
C
           NSM1 = NS - 1
           NEM1 = NE - 1
           NEM2 = NE - 2
C
           IF( ITRING . EQ . 0 ) THEN
           XV( 1 , I5 ) = 0.25 * ( XV( 1 , I1 ) + XV( 1 , I2 ) +
     .                             XV( 1 , I3 ) + XV( 1 , I4 ) )
           XV( 2 , I5 ) = 0.25 * ( XV( 2 , I1 ) + XV( 2 , I2 ) +
     .                             XV( 2 , I3 ) + XV( 2 , I4 ) )
           JV( 1 , I5 ) = 0
C
           DO 85 IR = 1 , MHQ
           HYDVVV( I5 , IR ) = 0.25 * ( HYDVVV( I1 , IR ) +
     .                                  HYDVVV( I2 , IR ) +
     .                                  HYDVVV( I3 , IR ) +
     .                                  HYDVVV( I4 , IR ) )
   85      CONTINUE
C
           JV( 2 , NV ) = N
           IF( JV( 2 , I2 ) . GT . 0 ) JV( 2 , I2 ) = N3
C
           DX = XV( 1 , I3 ) - XV( 1 , I1 )
           DXX = XV( 1 , I5 ) - XV( 1 , I1 )
           DY = XV( 2 , I3 ) - XV( 2 , I1 )
           DYY = XV( 2 , I5 ) - XV( 2 , I1 )
           XS( 3 , IT1 ) = .5 * ( DX * DYY - DXX * DY )
           DX = XV( 1 , I2 ) - XV( 1 , I3 )
           DXX = XV( 1 , I5 ) - XV( 1 , I3 )
           DY = XV( 2 , I2 ) - XV( 2 , I3 )
           DYY = XV( 2 , I5 ) - XV( 2 , I3 )
           XS( 3 , NSM1 ) = .5 * ( DX * DYY - DXX * DY )
           DX = XV( 1 , I4 ) - XV( 1 , I2 )
           DXX = XV( 1 , I5 ) - XV( 1 , I2 )
           DY = XV( 2 , I4 ) - XV( 2 , I2 )
           DYY = XV( 2 , I5 ) - XV( 2 , I2 )
           XS( 3 , NS ) = .5 * ( DX * DYY - DXX * DY )
           DX = XV( 1 , I1 ) - XV( 1 , I4 )
           DXX = XV( 1 , I5 ) - XV( 1 , I4 )
```

```
      DY = XV( 2 , I1 ) - XV( 2 , I4 )
      DYY = XV( 2 , I5 ) - XV( 2 , I4 )
      XS( 3 , IT2 ) = .5 * ( DX * DYY - DXX * DY )
C
      XS( 1 , IT1 ) = ( XV( 1 , I1 ) + XV( 1 , I3 ) +
     .                  XV( 1 , NV ) ) * THIRD
      XS( 1 , NSM1 ) = ( XV( 1 , I3 ) + XV( 1 , I2 ) +
     .                   XV( 1 , NV ) ) * THIRD
      XS( 2 , IT1 ) = ( XV( 2 , I1 ) + XV( 2 , I3 ) +
     .                  XV( 2 , NV ) ) * THIRD
      XS( 2 , NSM1 ) = ( XV( 2 , I3 ) + XV( 2 , I2 ) +
     .                   XV( 2 , NV ) ) * THIRD
C
      XS( 1 , NS ) = ( XV( 1 , I2 ) + XV( 1 , I4 ) +
     .                 XV( 1 , NV ) ) * THIRD
      XS( 1 , IT2 ) = ( XV( 1 , I4 ) + XV( 1 , I1 ) +
     .                  XV( 1 , NV ) ) * THIRD
      XS( 2 , NS ) = ( XV( 2 , I2 ) + XV( 2 , I4 ) +
     .                 XV( 2 , NV ) ) * THIRD
      XS( 2 , IT2 ) = ( XV( 2 , I4 ) + XV( 2 , I1 ) +
     .                  XV( 2 , NV ) ) * THIRD
C
      DO 94 IR = 1 , MHQ
      HYDV( IT1 , IR ) = ( HYDVVV( I1 , IR ) +
     .                     HYDVVV( I3 , IR ) +
     .                     HYDVVV( NV , IR ) ) * THIRD
      HYDV( NSM1 , IR ) = ( HYDVVV( I3 , IR ) +
     .                      HYDVVV( I2 , IR ) +
     .                      HYDVVV( NV , IR ) ) * THIRD
      HYDV( IT2 , IR ) = ( HYDVVV( I4 , IR ) +
     .                     HYDVVV( I1 , IR ) +
     .                     HYDVVV( NV , IR ) ) * THIRD
      HYDV( NS , IR ) = ( HYDVVV( I2 , IR ) +
     .                    HYDVVV( I4 , IR ) +
     .                    HYDVVV( NV , IR ) ) * THIRD
   94 CONTINUE
C
      HDUM              = 1. / ( HYDV( IT1 , 1 ) + 1.E-12 )
      HYDV( IT1 , 2 ) = HYDV( IT1 , 2 ) * HDUM
      HYDV( IT1 , 3 ) = HYDV( IT1 , 3 ) * HDUM
      HYDV( IT1 , 4 ) = ( HYDV( IT1 , 4 )  -
     .                 .5 * HYDV( IT1 , 1 ) *
     .   ( HYDV( IT1 , 2 ) * HYDV( IT1 , 2 ) +
     .     HYDV( IT1 , 3 ) * HYDV( IT1 , 3 ) ) ) *
     .                    ( HYDV( IT1 , 5 ) - 1. )
C
      HDUM              = 1. / ( HYDV( NSM1 , 1 ) + 1.E-12 )
      HYDV( NSM1 , 2 ) = HYDV( NSM1 , 2 ) * HDUM
      HYDV( NSM1 , 3 ) = HYDV( NSM1 , 3 ) * HDUM
      HYDV( NSM1 , 4 ) = ( HYDV( NSM1 , 4 )  -
     .                 .5 * HYDV( NSM1 , 1 ) *
     .   ( HYDV( NSM1 , 2 ) * HYDV( NSM1 , 2 ) +
     .     HYDV( NSM1 , 3 ) * HYDV( NSM1 , 3 ) ) ) *
     .                    ( HYDV( NSM1 , 5 ) - 1. )
C
      HDUM              = 1. / ( HYDV( IT2 , 1 ) + 1.E-12 )
      HYDV( IT2 , 2 ) = HYDV( IT2 , 2 ) * HDUM
      HYDV( IT2 , 3 ) = HYDV( IT2 , 3 ) * HDUM
      HYDV( IT2 , 4 ) = ( HYDV( IT2 , 4 )  -
     .                 .5 * HYDV( IT2 , 1 ) *
     .   ( HYDV( IT2 , 2 ) * HYDV( IT2 , 2 ) +
     .     HYDV( IT2 , 3 ) * HYDV( IT2 , 3 ) ) ) *
     .                    ( HYDV( IT2 , 5 ) - 1. )
C
      HDUM              = 1. / ( HYDV( NS , 1 ) + 1.E-12 )
      HYDV( NS , 2 ) = HYDV( NS , 2 ) * HDUM
      HYDV( NS , 3 ) = HYDV( NS , 3 ) * HDUM
      HYDV( NS , 4 ) = ( HYDV( NS , 4 )  -
     .               .5 * HYDV( NS , 1 ) *
     .   ( HYDV( NS , 2 ) * HYDV( NS , 2 ) +
     .     HYDV( NS , 3 ) * HYDV( NS , 3 ) ) ) *
     .                  ( HYDV( NS , 5 ) - 1. )
C
      SAREA( IT1 ) = 1. / XS( 3 , IT1 )
```

```
            SAREA( NSM1 ) = 1. / XS( 3 , NSM1 )
            SAREA( IT2 ) = 1. / XS( 3 , IT2 )
            SAREA( NS ) = 1. / XS( 3 , NS )
C
            DO 112 IR = 1 , 2
            RGRAD( NS , IR ) = RGRAD( IT2 , IR )
            RGRAD( NSM1 , IR ) = RGRAD( IT1 , IR )
            UGRAD( NS , IR ) = UGRAD( IT2 , IR )
            UGRAD( NSM1 , IR ) = UGRAD( IT1 , IR )
            VGRAD( NS , IR ) = VGRAD( IT2 , IR )
            VGRAD( NSM1 , IR ) = VGRAD( IT1 , IR )
            PGRAD( NS , IR ) = PGRAD( IT2 , IR )
            PGRAD( NSM1 , IR ) = PGRAD( IT1 , IR )
  112       CONTINUE
C
            KSDELT( NS ) = IDUMP
            KSDELT( NSM1 ) = IDUMP
            KSDELT( IT1 ) = IDUMP
            KSDELT( IT2 ) = IDUMP
C
            JEN( 1 ) = NAA
            JEN( 2 ) = NBB
            JEN( 3 ) = NCC
            JEN( 4 ) = NDD
            JEN( 5 ) = N
            JEN( 6 ) = N1
            JEN( 7 ) = N2
            JEN( 8 ) = N3
            JENN = 8
C
          ELSE
C
            XV( 1 , I5 ) = 0.5 * ( XV( 1 , I1 ) + XV( 1 , I2 ) )
            XV( 2 , I5 ) = 0.5 * ( XV( 2 , I1 ) + XV( 2 , I2 ) )
            JV( 1 , I5 ) = 0
C
            IF( IOSPCL . EQ . 1 . AND . IJE5 . EQ . 6 ) THEN
            ANGL = 1.570796327
            DXX = XV( 1 , I5 ) - 1.5
            IF( DXX . NE . 0. )  ANGL = ATAN2( XV( 2 , I5 ) , DXX )
            XV( 1 , I5 ) = COS( ANGL ) + 1.5
            XV( 2 , I5 ) = SIN( ANGL )
            END IF
C
            DO 80 IR = 1 , MHQ
            HYDVVV( I5 , IR ) = 0.5 * ( HYDVVV( I1 , IR ) +
     .                                  HYDVVV( I2 , IR ) )
   80       CONTINUE
C
            JV( 2 , I1 ) = - N
            JV( 2 , NV ) = - N3
C
            XSAREA = .5 * XS( 3 , IT2 )
            XS( 3 , IT2 ) = XSAREA
            XS( 3 , NS ) = XSAREA
C
            XS( 1 , NS ) = ( XV( 1 , I2 ) + XV( 1 , I4 ) +
     .                       XV( 1 , NV ) ) * THIRD
            XS( 1 , IT2 ) = ( XV( 1 , I4 ) + XV( 1 , I1 ) +
     .                        XV( 1 , NV ) ) * THIRD
            XS( 2 , NS ) = ( XV( 2 , I2 ) + XV( 2 , I4 ) +
     .                       XV( 2 , NV ) ) * THIRD
            XS( 2 , IT2 ) = ( XV( 2 , I4 ) + XV( 2 , I1 ) +
     .                        XV( 2 , NV ) ) * THIRD
C
            DO 92 IR = 1 , MHQ
            HYDV( IT2 , IR ) = ( HYDVVV( I4 , IR ) +
     .                           HYDVVV( I1 , IR ) +
     .                           HYDVVV( NV , IR ) ) * THIRD
            HYDV( NS , IR ) = ( HYDVVV( I2 , IR ) +
     .                          HYDVVV( I4 , IR ) +
     .                          HYDVVV( NV , IR ) ) * THIRD
   92       CONTINUE
C
```

```
            HDUM          = 1. / ( HYDV( IT2 , 1 ) + 1.E-12 )
            HYDV( IT2 , 2 ) = HYDV( IT2 , 2 ) * HDUM
            HYDV( IT2 , 3 ) = HYDV( IT2 , 3 ) * HDUM
            HYDV( IT2 , 4 ) = ( HYDV( IT2 , 4 )  -
     .                      .5 * HYDV( IT2 , 1 ) *
     .      ( HYDV( IT2 , 2 ) * HYDV( IT2 , 2 ) +
     .        HYDV( IT2 , 3 ) * HYDV( IT2 , 3 ) ) ) *
     .                         ( HYDV( IT2 , 5 ) - 1. )
C
            HDUM          = 1. / ( HYDV( NS , 1 ) + 1.E-12 )
            HYDV( NS , 2 ) = HYDV( NS , 2 ) * HDUM
            HYDV( NS , 3 ) = HYDV( NS , 3 ) * HDUM
            HYDV( NS , 4 ) = ( HYDV( NS , 4 )  -
     .                     .5 * HYDV( NS , 1 ) *
     .      ( HYDV( NS , 2 ) * HYDV( NS , 2 ) +
     .        HYDV( NS , 3 ) * HYDV( NS , 3 ) ) ) *
     .                        ( HYDV( NS , 5 ) - 1. )
C
            XSYREA = 1. / XSAREA
            SAREA( IT2 ) = XSYREA
            SAREA( NS ) = XSYREA
C
            DO 122 IR = 1 , 2
            RGRAD( NS , IR ) = RGRAD( IT2 , IR )
            UGRAD( NS , IR ) = UGRAD( IT2 , IR )
            VGRAD( NS , IR ) = VGRAD( IT2 , IR )
            PGRAD( NS , IR ) = PGRAD( IT2 , IR )
 122        CONTINUE
C
            KSDELT( NS ) = IDUMP
            KSDELT( IT2 ) = IDUMP
C
            JEN( 1 ) = NAA
            JEN( 2 ) = NDD
            JEN( 3 ) = N2
            JEN( 4 ) = N3
            JEN( 5 ) = N
            JENN = 5
C
            END IF
C
            DO 90 IENN = 1 , JENN
            IEN = JEN( IENN )
            JV1 = JE( 1 , IEN )
            JV2 = JE( 2 , IEN )
            AX = XV( 1 , JV2 ) - XV( 1 , JV1 )
            AY = XV( 2 , JV2 ) - XV( 2 , JV1 )
            XE( 1 , IEN ) = SQRT( AX * AX + AY * AY )
            XEREV = 1. / XE( 1 , IEN )
            XN( IEN ) = AY * XEREV
            YN( IEN ) = - AX * XEREV
            ISSR = JE( 4 , IEN )
            ISSL = JE( 3 , IEN )
C
            IJE5 = JE( 5 , IEN )
            IF( IJE5 . NE . 0 ) THEN
C
            AA = XV( 1 , JV2 ) - XV( 1 , JV1 )
            BB = XV( 2 , JV2 ) - XV( 2 , JV1 )
            XEL = XS( 1 , ISSL )
            YEL = XS( 2 , ISSL )
            CC = XEL - XV( 1 , JV1 )
            DD = YEL - XV( 2 , JV1 )
            EE = ( AA * CC + BB * DD ) * XEREV * XEREV
            XER = XV( 1 , JV1 ) + AA * EE
            YER = XV( 2 , JV1 ) + BB * EE
            AX = XER - XEL
            AY = YER - YEL
            XE( 2 , IEN ) = SQRT( AX * AX + AY * AY )
            XEREV = 1. / XE( 2 , IEN )
            XXN( IEN ) = AX * XEREV
            YYN( IEN ) = AY * XEREV
            XE( 2 , IEN ) = 2. * XE( 2 , IEN )
            XYMIDL( IEN ) = .5
```

```
                 XMIDL( IEN ) = XER
                 YMIDL( IEN ) = YER
      C
                 ELSE
      C
                 XER = XS( 1 , ISSR )
                 YER = XS( 2 , ISSR )
                 XEL = XS( 1 , ISSL )
                 YEL = XS( 2 , ISSL )
      C
                 AA = XV( 1 , JV2 ) - XV( 1 , JV1 )
                 BB = XV( 2 , JV2 ) - XV( 2 , JV1 )
                 CC = XEL - XER
                 DD = YEL - YER
                 ACA = XER - XV( 1 , JV1 )
                 DBD = YER - XV( 2 , JV1 )
                 EE = ( ACA * DD - DBD * CC ) / ( AA * DD - BB * CC )
                 XMIDL( IEN ) = XV( 1 , JV1 ) + AA * EE
                 YMIDL( IEN ) = XV( 2 , JV1 ) + BB * EE
      C
                 XEMID = XMIDL( IEN ) - XEL
                 YEMID = YMIDL( IEN ) - YEL
      C
                 AX = XER - XEL
                 AY = YER - YEL
                 XE( 2 , IEN ) = SQRT( AX * AX + AY * AY )
                 XEREV = 1. / XE( 2 , IEN )
                 XXN( IEN ) = AX * XEREV
                 YYN( IEN ) = AY * XEREV
      C
                 XYMIDL( IEN ) = SQRT( XEMID * XEMID + YEMID * YEMID ) * XEREV
      C
                 END IF
      C
       90      CONTINUE
      C
                 IF( IEROR.NE.0 ) THEN
                    WRITE (6,1000) N
                    IF( IEROR.EQ.2 ) WRITE (6,1002) I2, J1, IT1, I5
                    IF( IEROR.EQ.3 ) WRITE (6,1003) I2, J2, IT2, I5
                    STOP
                 END IF
      C
      C --- EXIT POINT FROM SUBROUTINE ------------------------------------
      C
      C     ------
            RETURN
      C     ------
      C
      C --- FORMATS -------------------------------------------------------
      C
       1000     FORMAT(/'0 TROUBLE WITH BOOKKEEPING DATA FOUND IN DISECT ',
           .            'FOR LINE ', I5///)
       1002     FORMAT(' DISECT  I2= ',I5,' J= ',I5,' IT1= ',I5,' I5= ',I5)
       1003     FORMAT(' DISECT  I2= ',I5,' J= ',I5,' IT2= ',I5,' I5= ',I5)
      C
      C     ---
            END
```

```
      SUBROUTINE DYNPTN( DAREA , NOFDIV , IDUMP , LTRIG )
C
C-------------------------------------------------------------I
C                                                             I
C      DYNPTN ADAPT THE GRID DYNAMICALLY, ADD VERTECES        I
C                                                             I
C-------------------------------------------------------------I
C
          IMPLICIT REAL (A-H,O-Z)
C
      include     'cmsh00.h'
      include     'chyd00.h'
      include     'cint00.h'
      include     'cphs10.h'
      include     'cphs20.h'
C
      INTEGER JTRIG(MEM),KTRIG(MEM),IRECNC(MEM)
      INTEGER JSE(MEM),JEE(MEM),IOFDVS(10),NOFDVS(10)
C
      EQUIVALENCE (UL,JTRIG)
      EQUIVALENCE (VR,KTRIG)
      EQUIVALENCE (VL,IRECNC)
      EQUIVALENCE (PR,JSE)
      EQUIVALENCE (PL,JEE)
C
       SMINVG = SAREVG * DAREA
       RMINVG = .7 * SMINVG
       DO 115 IS = 1 , NS
       JEE( IS ) = 0
  115  CONTINUE
C
       NSS = 0
       FLUXPP = .00001 * HYDMOM( 4 )
       FLUXUU = .00001 * HYDMOM( 2 )
       FLUXRR = .00001 * HYDMOM( 1 )
       DO 120 IS = 1 , NS
       PCRTRY = HYDFLX( IS , 4 ) - FLUXPP
       IPCRTR = SIGN( 1. , PCRTRY )
       UCRTRY = HYDFLX( IS , 2 ) - FLUXUU
       IUCRTR = SIGN( 1. , UCRTRY )
       RCRTRY = HYDFLX( IS , 1 ) - FLUXRR
       IRCRTR = SIGN( 1. , RCRTRY )
       IF( (
     .        IPCRTR . EQ . 1 . OR .
     .        IUCRTR . EQ . 1 . OR .
     .        IRCRTR . EQ . 1 ) . AND .
     .        KSDELT( IS ) . LT . IDUMP ) THEN
       KSDELT( IS ) = IDUMP
       JEE( IS ) = 1
       NSS = NSS + 1
       JTRIG( NSS ) = IS
       END IF
  120  CONTINUE
C
       DO 130 IS = 1 , NSS
       JSE( IS ) = JTRIG( IS )
  130  CONTINUE
C
       MSS = NSS
       DO 140 KDIV = 1 , NOFDIV
       ITRIG = 0
       DO 150 KS = 1 , NSS
C
       ISS = JSE( KS )
C
       DO 160 KR = 1 , 3
       IVV = JS( KR , ISS )
C
          IE = JV( 2 , IVV )
          IF( IE . GT . 0 ) THEN
C
          IV1 = JE( 1 , IE )
          IF( IV1 . EQ . IVV ) THEN
          ISI = JE( 3 , IE )
```

```
                ELSE
                ISI = JE( 4 , IE )
                END IF
                IS = ISI
C
750             CONTINUE
C
                JES = JEE( IS )
                XAS = XS( 3 , IS )
                IF( JES . EQ . 0 . AND . XAS . LT . SAREVG ) THEN
                ITRIG = ITRIG + 1
                KTRIG( ITRIG ) = IS
                KSDELT( IS ) = IDUMP
                JEE( IS ) = 1
                END IF
C
                DO 760 IR = 1 , 3
                JR = MOD( IR , 3 ) + 1
                IEA = IABS( JS( JR + 3 , IS ) )
                IF( IEA . EQ . IE ) THEN
                JJR = MOD( JR + 1 , 3 ) + 4
                IER = IABS( JS( JJR , IS ) )
C
                IV1 = JE( 1 , IER )
                IF( IV1 . EQ . IVV ) THEN
                ISR = JE( 3 , IER )
                ELSE
                ISR = JE( 4 , IER )
                END IF
                END IF
C
760             CONTINUE
C
                IF( ISR . NE . ISI ) THEN
                IS = ISR
                IE = IER
                GO TO 750
                END IF
C
                ELSE
C
                IE = - IE
                IV1 = JE( 1 , IE )
                IF( IV1 . EQ . IVV ) THEN
                ISI = JE( 3 , IE )
                ELSE
                ISI = JE( 4 , IE )
                END IF
                IS = ISI
                ISI = 0
                IIE = IE
C
650             CONTINUE
C
                JES = JEE( IS )
                XAS = XS( 3 , IS )
                IF( JES . EQ . 0 . AND . XAS . LT . SAREVG ) THEN
                ITRIG = ITRIG + 1
                KTRIG( ITRIG ) = IS
                KSDELT( IS ) = IDUMP
                JEE( IS ) = 1
                END IF
C
                DO 660 IR = 1 , 3
                JR = MOD( IR , 3 ) + 1
                IEA = IABS( JS( JR + 3 , IS ) )
                IF( IEA . EQ . IE ) THEN
                JJR = MOD( JR + 1 , 3 ) + 4
                IER = IABS( JS( JJR , IS ) )
C
                IV1 = JE( 1 , IER )
                IF( IV1 . EQ . IVV ) THEN
                ISR = JE( 3 , IER )
                ELSE
```

```
                ISR = JE( 4 , IER )
                END IF
                END IF
C
  660           CONTINUE
C
                IF( ISR . NE . ISI ) THEN
                IS = ISR
                IE = IER
                GO TO 650
                END IF
C
                END IF
  160   CONTINUE
C
  150   CONTINUE
C
        DO 170 IS = 1 , ITRIG
        JTRIG( IS + MSS ) = KTRIG( IS )
        JSE( IS ) = KTRIG( IS )
  170   CONTINUE
        NSS = ITRIG
        MSS = MSS + ITRIG
C
  140   CONTINUE
        NSS = MSS
C
        DO 300 KDIV = 1 , 1
        LTRIG = NSS
C
        DO 310 IS = 1 , NSS
        ISS = JTRIG( IS )
        XSAREA = XS( 3 , ISS )
        IF( XSAREA . GE . RMINVG ) THEN
C
        DO 335 IR = 4 , 6
        IE = IABS( JS( IR , ISS ) )
        IJE5 = JE( 5 , IE )
        IF( IJE5 . NE . 0 ) THEN
        JR2 = MOD( IR - 3 , 3 ) + 4
        IE2 = IABS( JS( JR2 , ISS ) )
        JR3 = MOD( IR - 2 , 3 ) + 4
        IE3 = IABS( JS( JR3 , ISS ) )
        XE1 = XE( 1 , IE )
        XE2 = XE( 1 , IE2 )
        XE3 = XE( 1 , IE3 )
        XEDIST = 1. / XE1
        YE2 = XE2 * XEDIST
        YE3 = XE3 * XEDIST
        ZE2 = ( YE2 - 1.5 ) * ( YE2 - .1 )
        ZE3 = ( YE3 - 1.5 ) * ( YE3 - .  )
        YY2 = XE1 * XE1 + XE2 * XE2 + .35 * XE1 * XE2 - XE3 * XE3
        YY3 = XE1 * XE1 + XE3 * XE3 + .35 * XE1 * XE3 - XE2 * XE2
        IF( ZE2 . LT . .0 . AND . ZE3 . LT . 0. . AND .
     .      YY2 . GT . 0. . AND . YY3 . GT . 0. ) THEN
        CALL DISECT ( IE , IDONE , IDUMP )
C
        LTRIG = LTRIG + 1
        JTRIG( LTRIG ) = NS
        KSDELT( NS ) = IDUMP
C
        END IF
        END IF
  335   CONTINUE
        END IF
  310   CONTINUE
C
        NSS = LTRIG
        IEDGE = 0
        NCOLOR = 0
C
        DO 295 IE = 1 , NE
        JSE( IE ) = 0
  295   CONTINUE
```

```
C
         DO 320 IS = 1 , NSS
         ISS = JTRIG( IS )
         XSAREA = XS( 3 , ISS )
C
         XXS = XS( 1 , ISS )
         YYS = XS( 2 , ISS )
         IZZ = 1
         IF( IWINDW . EQ . 1 ) THEN
         XXSS = - XXS * XXS + XXS + .75
         YYSS = - YYS * YYS + 1.
         IZZ = INT( SIGN( 1. , XXSS * YYSS ) )
         END IF
C
         IF( XSAREA . GT . RMINVG . AND . IZZ . EQ . 1 ) THEN
C
         DO 735 IR = 4 , 6
         IE = IABS( JS( IR , ISS ) )
         IF( JSE( IE ) . EQ . 0 ) THEN
         IEDGE = IEDGE + 1
         IRECNC( IEDGE ) = IE
         NCOLOR = NCOLOR + 1
         JEE( NCOLOR ) = IE
         JSE( IE ) = 1
         END IF
735      CONTINUE
C
         AREAXS = SAREA( ISS )
         IE1 = IABS( JS( 4 , ISS ) )
         XE1 = XE( 1 , IE1 )
         HD1 = AREAXS * XE1 * XE1
         IJE5 = JE( 5 , IE1 )
         IE2 = IABS( JS( 5 , ISS ) )
         XE2 = XE( 1 , IE2 )
         HD2 = AREAXS * XE2 * XE2
         IJE5 = IJE5 + JE( 5 , IE2 )
         IE3 = IABS( JS( 6 , ISS ) )
         XE3 = XE( 1 , IE3 )
         HD3 = AREAXS * XE3 * XE3
         IJE5 = IJE5 + JE( 5 , IE3 )
         RATIO = AMAX1( HD1 , HD2 , HD3 )
         IRATIO = 0
         IF( RATIO . LE . 7. . AND . IJE5 . EQ . 0 . AND .
     .                                XSAREA . GT . SMINVG ) IRATIO = 1
         IF( IJE5 . GT . 0 ) IRATIO = 2
C
         IF( IRATIO . EQ . 2 ) THEN
         IJE51 = JE( 5 , IE1 )
         IJE52 = JE( 5 , IE2 )
         IJE53 = JE( 5 , IE3 )
         IF( IJE51 . NE . 0 ) THEN
         IEDIST = IE1
         XE1 = XE( 1 , IE1 )
         XE2 = XE( 1 , IE2 )
         XE3 = XE( 1 , IE3 )
         END IF
         IF( IJE52 . NE . 0 ) THEN
         IEDIST = IE2
         XE1 = XE( 1 , IE2 )
         XE2 = XE( 1 , IE1 )
         XE3 = XE( 1 , IE3 )
         END IF
         IF( IJE53 . NE . 0 ) THEN
         IEDIST = IE3
         XE1 = XE( 1 , IE3 )
         XE2 = XE( 1 , IE2 )
         XE3 = XE( 1 , IE1 )
         END IF
         XEDIST = 1. / XE( 1 , IEDIST )
         YE2 = XE2 * XEDIST
         YE3 = XE3 * XEDIST
         ZE2 = ( YE2 - 1.5 ) * ( YE2 - .1 )
         ZE3 = ( YE3 - 1.5 ) * ( YE3 - .1 )
         YY2 = XE1 * XE1 + XE2 * XE2 + .35 * XE1 * XE2 - XE3 * XE3
```

```
      YY3 = XE1 * XE1 + XE3 * XE3 + .35 * XE1 * XE3 - XE2 * XE2
      IF( ZE2 . LT . .0 . AND . ZE3 . LT . 0. . AND .
     .    YY2 . GT . 0. . AND . YY3 . GT . 0. ) THEN
      CALL DISECT ( IEDICT , IDONE , IDUMP )
C
      LTRIG = LTRIG + 1
      JTRIG( LTRIG ) = NS
      KSDELT( NS ) = IDUMP
C
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE
      JSE( NE ) = 1
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE - 1
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE - 1
      JSE( NE - 1 ) = 1
C
      END IF
      END IF
C
      IF( IRATIO . EQ . 1 ) THEN
C
      CALL VERCEN( ISS )
      KSDELT( ISS ) = IDUMP
      LTRIG = LTRIG + 1
      JTRIG( LTRIG ) = NS - 1
      KSDELT( NS - 1 ) = IDUMP
C
      LTRIG = LTRIG + 1
      JTRIG( LTRIG ) = NS
      KSDELT( NS ) = IDUMP
C
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE
      JSE( NE ) = 1
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE - 1
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE - 1
      JSE( NE - 1 ) = 1
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE - 2
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE - 2
      JSE( NE - 2 ) = 1
C
      ELSE
C
      IDISCT = 0
      DO 545 KK = 4 , 6
      IEE = JS( KK , ISS )
      IEF = IABS( IEE )
      IJE55 = JE( 5 , IEF )
      IF( IJE55 . EQ . 0 ) THEN
      IF( IEE . GT . 0 ) THEN
      ISI = JE( 4 , IEE )
      ELSE
      ISI = JE( 3 , IEF )
      END IF
      AREAXS = SAREA( ISI )
      IE1 = IABS( JS( 4 , ISI ) )
      XE1 = XE( 1 , IE1 )
      IJE55 = JE( 5 , IE1 )
      HD1 = AREAXS * XE1 * XE1
      IE2 = IABS( JS( 5 , ISI ) )
      XE2 = XE( 1 , IE2 )
      IJE55 = IJE55 + JE( 5 , IE2 )
      HD2 = AREAXS * XE2 * XE2
      IE3 = IABS( JS( 6 , ISI ) )
```

```
      XE3 = XE( 1 , IE3 )
      IJE55 = IJE55 + JE( 5 , IE3 )
      HD3 = AREAXS * XE3 * XE3
      RATIO = AMAX1( HD1 , HD2 , HD3 )
      YSAREA = XS( 3 , ISI )
      IF( RATIO . LT . 7. . AND . YSAREA . GT . SMINVG . AND .
     .                            IJE55 . EQ . 0 ) THEN
      IDISCT = 1
      DO 435 IR = 4 , 6
      IE = IABS( JS( IR , ISI ) )
      IF( JSE( IE ) . EQ . 0 ) THEN
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = IE
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = IE
      JSE( IE ) = 1
      END IF
435   CONTINUE
      CALL VERCEN( ISI )
      KSDELT( ISI ) = IDUMP
      LTRIG = LTRIG + 1
      JTRIG( LTRIG ) = NS - 1
      KSDELT( NS - 1 ) = IDUMP
C
      LTRIG = LTRIG + 1
      JTRIG( LTRIG ) = NS
      KSDELT( NS ) = IDUMP
C
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE
      JSE( NE ) = 1
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE - 1
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE - 1
      JSE( NE - 1 ) = 1
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE - 2
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE - 2
      JSE( NE - 2 ) = 1
      END IF
      END IF
545   CONTINUE
C
      IF( IDISCT . EQ . 0 ) THEN
      IE1 = IABS( JS( 4 , ISS ) )
      XE1 = XE( 1 , IE1 )
      IE2 = IABS( JS( 5 , ISS ) )
      XE2 = XE( 1 , IE2 )
      IE3 = IABS( JS( 6 , ISS ) )
      XE3 = XE( 1 , IE3 )
      IEDIST = IE1
      XEDIST = XE1
      IF( XE2 . GT . XEDIST ) THEN
      XEDIST = XE2
      IEDIST = IE2
      END IF
      IF( XE3 . GT . XEDIST ) THEN
      XEDIST = XE3
      IEDIST = IE3
      END IF
      ISL = JE( 3 , IEDIST )
      ISR = JE( 4 , IEDIST )
      XSISL = XS( 3 , ISL )
      XSISR = XS( 3 , ISR )
      IJE5 = JE( 5 , IEDIST )
      IF( XSISL . GT . RMINVG . AND . XSISR . GT . RMINVG . AND .
     .    IJE5 . EQ . 0 . AND . IRATIO . NE . 2 )  THEN
      IF( ISS . NE . ISL ) THEN
      DO 345 IR = 4 , 6
      IE = IABS( JS( IR , ISL ) )
```

```
              IF( JSE( IE ) . EQ . 0 ) THEN
              IEDGE = IEDGE + 1
              IRECNC( IEDGE ) = IE
              NCOLOR = NCOLOR + 1
              JEE( NCOLOR ) = IE
              JSE( IE ) = 1
              END IF
  345         CONTINUE
              END IF
C
              IF( ISS . NE . ISR ) THEN
              DO 355 IR = 4 , 6
              IE = IABS( JS( IR , ISR ) )
              IF( JSE( IE ) . EQ . 0 ) THEN
              IEDGE = IEDGE + 1
              IRECNC( IEDGE ) = IE
              NCOLOR = NCOLOR + 1
              JEE( NCOLOR ) = IE
              JSE( IE ) = 1
              END IF
  355         CONTINUE
              END IF
C
              IDONE = 0
              CALL DISECT ( IEDIST , IDONE , IDUMP )
              IF( IDONE . EQ . 1 ) THEN
C
              LTRIG = LTRIG + 1
              JTRIG( LTRIG ) = NS
              KSDELT( NS ) = IDUMP
              LTRIG = LTRIG + 1
              JTRIG( LTRIG ) = NS - 1
              KSDELT( NS - 1 ) = IDUMP
C
              IEDGE = IEDGE + 1
              IRECNC( IEDGE ) = NE
              NCOLOR = NCOLOR + 1
              JEE( NCOLOR ) = NE
              JSE( NE ) = 1
              IEDGE = IEDGE + 1
              IRECNC( IEDGE ) = NE - 1
              NCOLOR = NCOLOR + 1
              JEE( NCOLOR ) = NE - 1
              JSE( NE - 1 ) = 1
              IEDGE = IEDGE + 1
              IRECNC( IEDGE ) = NE - 2
              NCOLOR = NCOLOR + 1
              JEE( NCOLOR ) = NE - 2
              JSE( NE - 2 ) = 1
              END IF
C
              END IF
              END IF
              END IF
              END IF
C
  320         CONTINUE
C
              DO 340 IEM = 1 , NCOLOR
              IE = JEE( IEM )
C
              ISL = JE( 3 , IE )
              YSAREA = XS( 3 , ISL )
              IJE5 = JE( 5 , IE )
              IF( YSAREA . GE . RMINVG . AND . IJE5 . NE . 0 ) THEN
              IE1 = IABS( JS( 4 , ISL ) )
              IE2 = IABS( JS( 5 , ISL ) )
              IE3 = IABS( JS( 6 , ISL ) )
              IJE51 = JE( 5 , IE1 )
              IJE52 = JE( 5 , IE2 )
              IJE53 = JE( 5 , IE3 )
              IF( IJE51 . NE . 0 ) THEN
              IEDIST = IE1
              XE1 = XE( 1 , IE1 )
```

```
      XE2 = XE( 1 , IE2 )
      XE3 = XE( 1 , IE3 )
      END IF
      IF( IJE52 . NE . 0 ) THEN
      IEDIST = IE2
      XE1 = XE( 1 , IE2 )
      XE2 = XE( 1 , IE1 )
      XE3 = XE( 1 , IE3 )
      END IF
      IF( IJE53 . NE . 0 ) THEN
      IEDIST = IE3
      XE1 = XE( 1 , IE3 )
      XE2 = XE( 1 , IE2 )
      XE3 = XE( 1 , IE1 )
      END IF
      XEDIST = 1. / XE( 1 , IEDIST )
      YE2 = XE2 * XEDIST
      YE3 = XE3 * XEDIST
      ZE2 = ( YE2 - 1.5 ) * ( YE2 - .1 )
      ZE3 = ( YE3 - 1.5 ) * ( YE3 - .1 )
      YY2 = XE1 * XE1 + XE2 * XE2 + .35 * XE1 * XE2 - XE3 * XE3
      YY3 = XE1 * XE1 + XE3 * XE3 + .35 * XE1 * XE3 - XE2 * XE2
      IF( ZE2 . LT . .0 . AND . ZE3 . LT . 0. . AND .
     .    YY2 . GT . 0. . AND . YY3 . GT . 0. ) THEN
      CALL DISECT ( IEDIST , IDONE , IDUMP )
C
      LTRIG = LTRIG + 1
      JTRIG( LTRIG ) = NS
      KSDELT( NS ) = IDUMP
C
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE
      JSE( NE ) = 1
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE - 1
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE - 1
      JSE( NE - 1 ) = 1
C
       ELSE
C
      IEDIST = IE1
      XEDIST = XE1
      IF( XE2 . GT . XEDIST ) THEN
      XEDIST = XE2
      IEDIST = IE2
      END IF
      IF( XE3 . GT . XEDIST ) THEN
      XEDIST = XE3
      IEDIST = IE3
      END IF
      ISL = JE( 3 , IEDIST )
      ISR = JE( 4 , IEDIST )
      XSISL = XS( 3 , ISL )
      XSISR = XS( 3 , ISR )
      IJE5 = JE( 5 , IEDIST )
      IF( XSISL . GT . RMINVG . AND . XSISR . GT . RMINVG . AND .
     .                                   IJE5 . EQ . 0 )  THEN
      DO 645 IR = 4 , 6
      IE = IABS( JS( IR , ISL ) )
      IF( JSE( IE ) . EQ . 0 ) THEN
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = IE
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = IE
      JSE( IE ) = 1
      END IF
  645 CONTINUE
      DO 655 IR = 4 , 6
      IE = IABS( JS( IR , ISR ) )
      IF( JSE( IE ) . EQ . 0 ) THEN
      IEDGE = IEDGE + 1
```

```
            IRECNC( IEDGE ) = IE
            NCOLOR = NCOLOR + 1
            JEE( NCOLOR ) = IE
            JSE( IE ) = 1
            END IF
  655       CONTINUE
C
            IDONE = 0
            CALL DISECT ( IEDIST , IDONE , IDUMP )
            IF( IDONE . EQ . 1 ) THEN
C
            LTRIG = LTRIG + 1
            JTRIG( LTRIG ) = NS
            KSDELT( NS ) = IDUMP
            LTRIG = LTRIG + 1
            JTRIG( LTRIG ) = NS - 1
            KSDELT( NS - 1 ) = IDUMP
C
            IEDGE = IEDGE + 1
            IRECNC( IEDGE ) = NE
            NCOLOR = NCOLOR + 1
            JEE( NCOLOR ) = NE
            JSE( NE ) = 1
            IEDGE = IEDGE + 1
            IRECNC( IEDGE ) = NE - 1
            NCOLOR = NCOLOR + 1
            JEE( NCOLOR ) = NE - 1
            JSE( NE - 1 ) = 1
            IEDGE = IEDGE + 1
            IRECNC( IEDGE ) = NE - 2
            NCOLOR = NCOLOR + 1
            JEE( NCOLOR ) = NE - 2
            JSE( NE - 2 ) = 1
            END IF
C
            END IF
            END IF
            END IF
  340       CONTINUE
C
            NSS = LTRIG
C
            DO 370 IEM = 1 , NCOLOR
            IE = JEE( IEM )
            CALL RECNC( IE , IDONE , ITL , ITR , JA , JB , JC , JD )
            CALL RECNC( JA , JADONE , ITL , ITR , JAA , JAB , JAC , JAD )
            CALL RECNC( JB , JBDONE , ITL , ITR , JBA , JBB , JBC , JBD )
            CALL RECNC( JC , JCDONE , ITL , ITR , JCA , JCB , JCC , JCD )
            CALL RECNC( JD , JDDONE , ITL , ITR , JDA , JDB , JDC , JDD )
  370       CONTINUE
C
  300       CONTINUE
C
          NVECE = NE / MBL
          NREME = NE - NVECE * MBL
          NVECS = NS / MBL
          NREMS = NS - NVECS * MBL
          NVECV = NV / MBL
          NREMV = NV - NVECV * MBL
C
          DO 400 INE = 1 , NVECE
          NOFVEE( INE ) = MBL
  400     CONTINUE
          NVEEE = NVECE
          IF( NREME . GT . 0 ) THEN
          NVEEE = NVECE + 1
          NOFVEE( NVEEE ) = NREME
          END IF
C
          DO 410 INS = 1 , NVECS
          NOFVES( INS ) = MBL
  410     CONTINUE
          NVEES = NVECS
          IF( NREMS . GT . 0 ) THEN
```

```
         NVEES = NVECS + 1
         NOFVES( NVEES ) = NREMS
         END IF
C
         DO 420 INV = 1 , NVECV
         NOFVEV( INV ) = MBL
  420    CONTINUE
         NVEEV = NVECV
         IF( NREMV . GT . 0 ) THEN
         NVEEV = NVECV + 1
         NOFVEV( NVEEV ) = NREMV
         END IF
C
          PRINT*,NV,NE,NS
C
C --- EXIT POINT FROM SUBROUTINE -------------------------------------
C
C        ------
         RETURN
C        ------
C
C        ---
         END
```

Thu Jul 1 14:16:08 1993 adaphd.f SUBROUTINE DYYPTN

```
         SUBROUTINE DYYPTN( DAREA , NOFDIV , IDUMP , LTRIG )
C
C-------------------------------------------------------------------I
C                                                                   I
C         DYYPTN ADAPT THE GRID DYNAMICALLY, ADD VERTECES           I
C         SUB DIVIDE THE TRIANGLE THAT WERE FLAGED IN DYNPTN        I
C                                                                   I
C-------------------------------------------------------------------I
C
             IMPLICIT REAL (A-H,O-Z)
C
         include     'cmsh00.h'
         include     'chyd00.h'
         include     'cint00.h'
         include     'cphs10.h'
         include     'cphs20.h'
C
         INTEGER JTRIG(MEM),KTRIG(MEM),IRECNC(MEM)
         INTEGER JSE(MEM),JEE(MEM),IOFDVS(10),NOFDVS(10)
C
         EQUIVALENCE (UL,JTRIG)
         EQUIVALENCE (VR,KTRIG)
         EQUIVALENCE (VL,IRECNC)
         EQUIVALENCE (PR,JSE)
         EQUIVALENCE (PL,JEE)
C
          SMINVG = SAREVG * DAREA
          AMINVG = SAREVG * THIRD
          RMINVG = .7 * SMINVG
          DO 115 IS = 1 , NS
          JEE( IS ) = 0
  115     CONTINUE
          MSS = 0
          NSS = LTRIG
C
          DO 140 KDIV = 1 , NOFDIV
          ITRIG = 0
          DO 150 KS = 1 , NSS
C
          ISS = JTRIG( KS )
          IF( ISS . NE . 0 ) THEN
C
          DO 160 KR = 1 , 3
          IVV = JS( KR , ISS )
C
             IE = JV( 2 , IVV )
             IF( IE . GT . 0 ) THEN
C
```

```
      IV1 = JE( 1 , IE )
      IF( IV1 . EQ . IVV ) THEN
      ISI = JE( 3 , IE )
      ELSE
      ISI = JE( 4 , IE )
      END IF
      IS = ISI
C
750   CONTINUE
C
      JES = JEE( IS )
      XAS = XS( 3 , IS )
      IF( JES . EQ . 0 . AND . XAS . GT . RMINVG ) THEN
      ITRIG = ITRIG + 1
      KTRIG( ITRIG ) = IS
      KSDELT( IS ) = IDUMP
      JEE( IS ) = 1
      END IF
C
      DO 760 IR = 1 , 3
      JR = MOD( IR , 3 ) + 1
      IEA = IABS( JS( JR + 3 , IS ) )
      IF( IEA . EQ . IE ) THEN
      JJR = MOD( JR + 1 , 3 ) + 4
      IER = IABS( JS( JJR , IS ) )
C
      IV1 = JE( 1 , IER )
      IF( IV1 . EQ . IVV ) THEN
      ISR = JE( 3 , IER )
      ELSE
      ISR = JE( 4 , IER )
      END IF
      END IF
C
760   CONTINUE
C
      IF( ISR . NE . ISI ) THEN
      IS = ISR
      IE = IER
      GO TO 750
      END IF
C
      ELSE
C
      IE = - IE
      IV1 = JE( 1 , IE )
      IF( IV1 . EQ . IVV ) THEN
      ISI = JE( 3 , IE )
      ELSE
      ISI = JE( 4 , IE )
      END IF
      IS = ISI
      ISI = 0
      IIE = IE
C
650   CONTINUE
C
      JES = JEE( IS )
      XAS = XS( 3 , IS )
      IF( JES . EQ . 0 . AND . XAS . GT . RMINVG ) THEN
      ITRIG = ITRIG + 1
      KTRIG( ITRIG ) = IS
      KSDELT( IS ) = IDUMP
      JEE( IS ) = 1
      END IF
C
      DO 660 IR = 1 , 3
      JR = MOD( IR , 3 ) + 1
      IEA = IABS( JS( JR + 3 , IS ) )
      IF( IEA . EQ . IE ) THEN
      JJR = MOD( JR + 1 , 3 ) + 4
      IER = IABS( JS( JJR , IS ) )
C
      IV1 = JE( 1 , IER )
```

```
            IF( IV1 . EQ . IVV ) THEN
            ISR = JE( 3 , IER )
            ELSE
            ISR = JE( 4 , IER )
            END IF
            END IF
C
660         CONTINUE
C
            IF( ISR . NE . ISI ) THEN
            IS = ISR
            IE = IER
            GO TO 650
            END IF
C
            END IF
  160   CONTINUE
C
            END IF
  150   CONTINUE
C
        DO 170 IS = 1 , ITRIG
        JTRIG( IS + MSS ) = KTRIG( IS )
  170   CONTINUE
        NSS = ITRIG
        MSS = MSS + ITRIG
C
  140   CONTINUE
        NSS = MSS
C
        DO 300 KDIV = 1 , 1
        LTRIG = NSS
C
        DO 310 IS = 1 , NSS
        ISS = JTRIG( IS )
        XSAREA = XS( 3 , ISS )
        IF( XSAREA . GE . RMINVG ) THEN
C
        DO 335 IR = 4 , 6
        IE = IABS( JS( IR , ISS ) )
        IJE5 = JE( 5 , IE )
        IF( IJE5 . NE . 0 ) THEN
        JR2 = MOD( IR - 3 , 3 ) + 4
        IE2 = IABS( JS( JR2 , ISS ) )
        JR3 = MOD( IR - 2 , 3 ) + 4
        IE3 = IABS( JS( JR3 , ISS ) )
        XE1 = XE( 1 , IE )
        XE2 = XE( 1 , IE2 )
        XE3 = XE( 1 , IE3 )
        XEDIST = 1. / XE1
        YE2 = XE2 * XEDIST
        YE3 = XE3 * XEDIST
        ZE2 = ( YE2 - 1.5 ) * ( YE2 - .1 )
        ZE3 = ( YE3 - 1.5 ) * ( YE3 - .1 )
        YY2 = XE1 * XE1 + XE2 * XE2 + .35 * XE1 * XE2 - XE3 * XE3
        YY3 = XE1 * XE1 + XE3 * XE3 + .35 * XE1 * XE3 - XE2 * XE2
        IF( ZE2 . LT . .0 . AND . ZE3 . LT . 0. . AND .
     .      YY2 . GT . 0. . AND . YY3 . GT . 0. ) THEN
        CALL DISECT ( IE , IDONE , IDUMP )
C
        LTRIG = LTRIG + 1
        JTRIG( LTRIG ) = NS
        KSDELT( NS ) = IDUMP
C
        END IF
        END IF
  335   CONTINUE
        END IF
  310   CONTINUE
C
        NSS = LTRIG
        IEDGE = 0
        NCOLOR = 0
C
```

```
            DO 295 IE = 1 , NE
            JSE( IE ) = 0
  295       CONTINUE
C
            DO 320 IS = 1 , NSS
            ISS = JTRIG( IS )
            XSAREA = XS( 3 , ISS )
C
            IF( XSAREA . GT . RMINVG ) THEN
C
            DO 735 IR = 4 , 6
            IE = IABS( JS( IR , ISS ) )
            IF( JSE( IE ) . EQ . 0 ) THEN
            IEDGE = IEDGE + 1
            IRECNC( IEDGE ) = IE
            NCOLOR = NCOLOR + 1
            JEE( NCOLOR ) = IE
            JSE( IE ) = 1
            END IF
  735       CONTINUE
C
            AREAXS = SAREA( ISS )
            IE1 = IABS( JS( 4 , ISS ) )
            XE1 = XE( 1 , IE1 )
            HD1 = AREAXS * XE1 * XE1
            IJE5 = JE( 5 , IE1 )
            IE2 = IABS( JS( 5 , ISS ) )
            XE2 = XE( 1 , IE2 )
            HD2 = AREAXS * XE2 * XE2
            IJE5 = IJE5 + JE( 5 , IE2 )
            IE3 = IABS( JS( 6 , ISS ) )
            XE3 = XE( 1 , IE3 )
            HD3 = AREAXS * XE3 * XE3
            IJE5 = IJE5 + JE( 5 , IE3 )
            RATIO = AMAX1( HD1 , HD2 , HD3 )
            IRATIO = 0
            IF( RATIO . LE . 7. . AND . IJE5 . EQ . 0 . AND .
     .                           XSAREA . GT . SMINVG ) IRATIO = 1
            IF( IJE5 . GT . 0 ) IRATIO = 2
C
            IF( IRATIO . EQ . 2 ) THEN
            IJE51 = JE( 5 , IE1 )
            IJE52 = JE( 5 , IE2 )
            IJE53 = JE( 5 , IE3 )
            IF( IJE51 . NE . 0 ) THEN
            IEDIST = IE1
            XE1 = XE( 1 , IE1 )
            XE2 = XE( 1 , IE2 )
            XE3 = XE( 1 , IE3 )
            END IF
            IF( IJE52 . NE . 0 ) THEN
            IEDIST = IE2
            XE1 = XE( 1 , IE2 )
            XE2 = XE( 1 , IE1 )
            XE3 = XE( 1 , IE3 )
            END IF
            IF( IJE53 . NE . 0 ) THEN
            IEDIST = IE3
            XE1 = XE( 1 , IE3 )
            XE2 = XE( 1 , IE2 )
            XE3 = XE( 1 , IE1 )
            END IF
            XEDIST = 1. / XE( 1 , IEDIST )
            YE2 = XE2 * XEDIST
            YE3 = XE3 * XEDIST
            ZE2 = ( YE2 - 1.5 ) * ( YE2 - .1 )
            ZE3 = ( YE3 - 1.5 ) * ( YE3 - .1 )
            YY2 = XE1 * XE1 + XE2 * XE2 + .35 * XE1 * XE2 - XE3 * XE3
            YY3 = XE1 * XE1 + XE3 * XE3 + .35 * XE1 * XE3 - XE2 * XE2
            IF( ZE2 . LT . .0 . AND . ZE3 . LT . 0. . AND .
     .         YY2 . GT . 0. . AND . YY3 . GT . 0. ) THEN
            CALL DISECT ( IEDIST , IDONE , IDUMP )
C
            LTRIG = LTRIG + 1
```

```
      JTRIG( LTRIG ) = NS
      KSDELT( NS ) = IDUMP
C
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE
      JSE( NE ) = 1
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE - 1
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE - 1
      JSE( NE - 1 ) = 1
C
      END IF
      END IF
C
      IF( IRATIO . EQ . 1 ) THEN
C
      CALL VERCEN( ISS )
      KSDELT( ISS ) = IDUMP
      LTRIG = LTRIG + 1
      JTRIG( LTRIG ) = NS - 1
      KSDELT( NS - 1 ) = IDUMP
C
      LTRIG = LTRIG + 1
      JTRIG( LTRIG ) = NS
      KSDELT( NS ) = IDUMP
C
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE
      JSE( NE ) = 1
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE - 1
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE - 1
      JSE( NE - 1 ) = 1
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE - 2
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE - 2
      JSE( NE - 2 ) = 1
C
      ELSE
C
      IDISCT = 0
      DO 545 KK = 4 , 6
      IEE = JS( KK , ISS )
      IEF = IABS( IEE )
      IJE55 = JE( 5 , IEF )
      IF( IJE55 . EQ . 0 ) THEN
      IF( IEE . GT . 0 ) THEN
      ISI = JE( 4 , IEE )
      ELSE
      ISI = JE( 3 , IEF )
      END IF
      AREAXS = SAREA( ISI )
      IE1 = IABS( JS( 4 , ISI ) )
      XE1 = XE( 1 , IE1 )
      IJE55 = JE( 5 , IE1 )
      HD1 = AREAXS * XE1 * XE1
      IE2 = IABS( JS( 5 , ISI ) )
      XE2 = XE( 1 , IE2 )
      IJE55 = IJE55 + JE( 5 , IE2 )
      HD2 = AREAXS * XE2 * XE2
      IE3 = IABS( JS( 6 , ISI ) )
      XE3 = XE( 1 , IE3 )
      IJE55 = IJE55 + JE( 5 , IE3 )
      HD3 = AREAXS * XE3 * XE3
      RATIO = AMAX1( HD1 , HD2 , HD3 )
      YSAREA = XS( 3 , ISI )
      IF( RATIO . LT . 7. . AND . YSAREA . GT . SMINVG . AND .
```

```
     .                                   IJE55 . EQ . 0 ) THEN
        IDISCT = 1
        DO 435 IR = 4 , 6
        IE = IABS( JS( IR , ISI ) )
        IF( JSE( IE ) . EQ . 0 ) THEN
        IEDGE = IEDGE + 1
        IRECNC( IEDGE ) = IE
        NCOLOR = NCOLOR + 1
        JEE( NCOLOR ) = IE
        JSE( IE ) = 1
        END IF
  435   CONTINUE
        CALL VERCEN( ISI )
        KSDELT( ISI ) = IDUMP
        LTRIG = LTRIG + 1
        JTRIG( LTRIG ) = NS - 1
        KSDELT( NS - 1 ) = IDUMP
C
        LTRIG = LTRIG + 1
        JTRIG( LTRIG ) = NS
        KSDELT( NS ) = IDUMP
C
        IEDGE = IEDGE + 1
        IRECNC( IEDGE ) = NE
        NCOLOR = NCOLOR + 1
        JEE( NCOLOR ) = NE
        JSE( NE ) = 1
        IEDGE = IEDGE + 1
        IRECNC( IEDGE ) = NE - 1
        NCOLOR = NCOLOR + 1
        JEE( NCOLOR ) = NE - 1
        JSE( NE - 1 ) = 1
        IEDGE = IEDGE + 1
        IRECNC( IEDGE ) = NE - 2
        NCOLOR = NCOLOR + 1
        JEE( NCOLOR ) = NE - 2
        JSE( NE - 2 ) = 1
        END IF
        END IF
  545   CONTINUE
C
        IF( IDISCT . EQ . 0 ) THEN
        IE1 = IABS( JS( 4 , ISS ) )
        XE1 = XE( 1 , IE1 )
        IE2 = IABS( JS( 5 , ISS ) )
        XE2 = XE( 1 , IE2 )
        IE3 = IABS( JS( 6 , ISS ) )
        XE3 = XE( 1 , IE3 )
        IEDIST = IE1
        XEDIST = XE1
        IF( XE2 . GT . XEDIST ) THEN
        XEDIST = XE2
        IEDIST = IE2
        END IF
        IF( XE3 . GT . XEDIST ) THEN
        XEDIST = XE3
        IEDIST = IE3
        END IF
        ISL = JE( 3 , IEDIST )
        ISR = JE( 4 , IEDIST )
        XSISL = XS( 3 , ISL )
        XSISR = XS( 3 , ISR )
        IJE5 = JE( 5 , IEDIST )
        IF( XSISL . GT . RMINVG . AND . XSISR . GT . RMINVG . AND .
     .      IJE5 . EQ . 0 . AND . IRATIO . NE . 2 )  THEN
        IF( ISS . NE . ISL ) THEN
        DO 345 IR = 4 , 6
        IE = IABS( JS( IR , ISL ) )
        IF( JSE( IE ) . EQ . 0 ) THEN
        IEDGE = IEDGE + 1
        IRECNC( IEDGE ) = IE
        NCOLOR = NCOLOR + 1
        JEE( NCOLOR ) = IE
        JSE( IE ) = 1
```

```
      END IF
  345 CONTINUE
      END IF
C
      IF( ISS . NE . ISR ) THEN
      DO 355 IR = 4 , 6
      IE = IABS( JS( IR , ISR ) )
      IF( JSE( IE ) . EQ . 0 ) THEN
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = IE
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = IE
      JSE( IE ) = 1
      END IF
  355 CONTINUE
      END IF
C
      IDONE = 0
      CALL DISECT ( IEDIST , IDONE , IDUMP )
      IF( IDONE . EQ . 1 ) THEN
C
      LTRIG = LTRIG + 1
      JTRIG( LTRIG ) = NS
      KSDELT( NS ) = IDUMP
      LTRIG = LTRIG + 1
      JTRIG( LTRIG ) = NS - 1
      KSDELT( NS - 1 ) = IDUMP
C
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE
      JSE( NE ) = 1
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE - 1
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE - 1
      JSE( NE - 1 ) = 1
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE - 2
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE - 2
      JSE( NE - 2 ) = 1
      END IF
C
      END IF
      END IF
      END IF
      END IF
C
  320 CONTINUE
C
      DO 340 IEM = 1 , NCOLOR
      IE = JEE( IEM )
C
      ISL = JE( 3 , IE )
      YSAREA = XS( 3 , ISL )
      IJE5 = JE( 5 , IE )
      IF( YSAREA . GE . RMINVG . AND . IJE5 . NE . 0 ) THEN
      IE1 = IABS( JS( 4 , ISL ) )
      IE2 = IABS( JS( 5 , ISL ) )
      IE3 = IABS( JS( 6 , ISL ) )
      IJE51 = JE( 5 , IE1 )
      IJE52 = JE( 5 , IE2 )
      IJE53 = JE( 5 , IE3 )
      IF( IJE51 . NE . 0 ) THEN
      IEDIST = IE1
      XE1 = XE( 1 , IE1 )
      XE2 = XE( 1 , IE2 )
      XE3 = XE( 1 , IE3 )
      END IF
      IF( IJE52 . NE . 0 ) THEN
      IEDIST = IE2
      XE1 = XE( 1 , IE2 )
```

```
            XE2 = XE( 1 , IE1 )
            XE3 = XF( 1 , IE3 )
            END IF
            IF( IJE53 . NE . 0 ) THEN
            IEDIST = IE3
            XE1 = XE( 1 , IE3 )
            XE2 = XE( 1 , IE2 )
            XE3 = XE( 1 , IE1 )
            END IF
            XEDIST = 1. / XE( 1 , IEDIST )
            YE2 = XE2 * XEDIST
            YE3 = XE3 * XEDIST
            ZE2 = ( YE2 - 1.5 ) * ( YE2 - .1 )
            ZE3 = ( YE3 - 1.5 ) * ( YE3 - .1 )
            YY2 = XE1 * XE1 + XE2 * XE2 + .35 * XE1 * XE2 - XE3 * XE3
            YY3 = XE1 * XE1 + XE3 * XE3 + .35 * XE1 * XE3 - XE2 * XE2
            IF( ZE2 . LT . .0 . AND . ZE3 . LT . 0. . AND .
     .          YY2 . GT . 0. . AND . YY3 . GT . 0. ) THEN
            CALL DISECT ( IEDIST , IDONE , IDUMP )
      C
            LTRIG = LTRIG + 1
            JTRIG( LTRIG ) = NS
            KSDELT( NS ) = IDUMP
      C
            IEDGE = IEDGE + 1
            IRECNC( IEDGE ) = NE
            NCOLOR = NCOLOR + 1
            JEE( NCOLOR ) = NE
            JSE( NE ) = 1
            IEDGE = IEDGE + 1
            IRECNC( IEDGE ) = NE - 1
            NCOLOR = NCOLOR + 1
            JEE( NCOLOR ) = NE - 1
            JSE( NE - 1 ) = 1
      C
             ELSE
      C
            IEDIST = IE1
            XEDIST = XE1
            IF( XE2 . GT . XEDIST ) THEN
            XEDIST = XE2
            IEDIST = IE2
            END IF
            IF( XE3 . GT . XEDIST ) THEN
            XEDIST = XE3
            IEDIST = IE3
            END IF
            ISL = JE( 3 , IEDIST )
            ISR = JE( 4 , IEDIST )
            XSISL = XS( 3 , ISL )
            XSISR = XS( 3 , ISR )
            IJE5 = JE( 5 , IEDIST )
            IF( XSISL . GT . RMINVG . AND . XSISR . GT . RMINVG . AND .
     .                                         IJE5 . EQ . 0 )  THEN
            DO 645 IR = 4 , 6
            IE = IABS( JS( IR , ISL ) )
            IF( JSE( IE ) . EQ . 0 ) THEN
            IEDGE = IEDGE + 1
            IRECNC( IEDGE ) = IE
            NCOLOR = NCOLOR + 1
            JEE( NCOLOR ) = IE
            JSE( IE ) = 1
            END IF
      645   CONTINUE
            DO 655 IR = 4 , 6
            IE = IABS( JS( IR , ISR ) )
            IF( JSE( IE ) . EQ . 0 ) THEN
            IEDGE = IEDGE + 1
            IRECNC( IEDGE ) = IE
            NCOLOR = NCOLOR + 1
            JEE( NCOLOR ) = IE
            JSE( IE ) = 1
            END IF
      655   CONTINUE
```

```
C
            IDONE = 0
            CALL DISECT ( IEDIST , IDONE , IDUMP )
            IF( IDONE . EQ . 1 ) THEN
C
            LTRIG = LTRIG + 1
            JTRIG( LTRIG ) = NS
            KSDELT( NS ) = IDUMP
            LTRIG = LTRIG + 1
            JTRIG( LTRIG ) = NS - 1
            KSDELT( NS - 1 ) = IDUMP
C
            IEDGE = IEDGE + 1
            IRECNC( IEDGE ) = NE
            NCOLOR = NCOLOR + 1
            JEE( NCOLOR ) = NE
            JSE( NE ) = 1
            IEDGE = IEDGE + 1
            IRECNC( IEDGE ) = NE - 1
            NCOLOR = NCOLOR + 1
            JEE( NCOLOR ) = NE - 1
            JSE( NE - 1 ) = 1
            IEDGE = IEDGE + 1
            IRECNC( IEDGE ) = NE - 2
            NCOLOR = NCOLOR + 1
            JEE( NCOLOR ) = NE - 2
            JSE( NE - 2 ) = 1
            END IF
C
            END IF
            END IF
            END IF
 340        CONTINUE
C
            NSS = LTRIG
C
            DO 370 IEM = 1 , NCOLOR
            IE = JEE( IEM )
            CALL RECNC( IE , IDONE , ITL , ITR , JA , JB , JC , JD )
            CALL RECNC( JA , JADONE , ITL , ITR , JAA , JAB , JAC , JAD )
            CALL RECNC( JB , JBDONE , ITL , ITR , JBA , JBB , JBC , JBD )
            CALL RECNC( JC , JCDONE , ITL , ITR , JCA , JCB , JCC , JCD )
            CALL RECNC( JD , JDDONE , ITL , ITR , JDA , JDB , JDC , JDD )
 370        CONTINUE
C
 300        CONTINUE
C
          NVECE = NE / MBL
          NREME = NE - NVECE * MBL
          NVECS = NS / MBL
          NREMS = NS - NVECS * MBL
          NVECV = NV / MBL
          NREMV = NV - NVECV * MBL
C
          DO 400 INE = 1 , NVECE
          NOFVEE( INE ) = MBL
 400      CONTINUE
          NVEEE = NVECE
          IF( NREME . GT . 0 ) THEN
          NVEEE = NVECE + 1
          NOFVEE( NVEEE ) = NREME
          END IF
C
          DO 410 INS = 1 , NVECS
          NOFVES( INS ) = MBL
 410      CONTINUE
          NVEES = NVECS
          IF( NREMS . GT . 0 ) THEN
          NVEES = NVECS + 1
          NOFVES( NVEES ) = NREMS
          END IF
C
          DO 420 INV = 1 , NVECV
          NOFVEV( INV ) = MBL
```

```
 420  CONTINUE
      NVEEV = NVECV
      IF( NREMV . GT . 0 ) THEN
      NVEEV = NVECV + 1
      NOFVEV( NVEEV ) = NREMV
      END IF
C
       PRINT*,NV,NE,NS
C
C --- EXIT POINT FROM SUBROUTINE -------------------------------------
C
C     ------
      RETURN
C     ------
C
C     ---
      END
```



```
      SUBROUTINE INTPTN( DAREA , NOFDIV , IDUMP , LTRIG )
C
C---------------------------------------------------------------|
C                                                               I
C      INTPTN ADAPT THE GRID DYNAMICALLY, ADD VERTECES          I
C      SUB DIVIDE TO REFINE AT THE INITIAL STAGE OF THE SIMULATION I
C                                                               I
C---------------------------------------------------------------I
C
          IMPLICIT REAL (A-H,O-Z)
C
      include     'cmsh00.h'
      include     'chyd00.h'
      include     'cint00.h'
      include     'cphs10.h'
      include     'cphs20.h'
C
      INTEGER JTRIG(MEM),KTRIG(MEM),IRECNC(MEM)
      INTEGER JSE(MEM),JEE(MEM),IOFDVS(10),NOFDVS(10)
C
      EQUIVALENCE (UL,JTRIG)
      EQUIVALENCE (VR,KTRIG)
      EQUIVALENCE (VL,IRECNC)
      EQUIVALENCE (PR,JSE)
      EQUIVALENCE (PL,JEE)
C
       SMINVG = SAREVG * DAREA
       RMINVG = .7 * SMINVG
C
       DO 115 IS = 1 , NS
       JEE( IS ) = 0
  115  CONTINUE
C
       NSS = 0
       DO 120 IS = 1 , NS
C       DO 120 IR = 4 , 6
C        IE = IABS( JS( IR , IS ) )
C        IJE5 = JE( 5 , IE )
       XSS = XS( 1 , IS )
       IF( XSS . GT . -.05 . AND . XSS . LT . .05 . AND .
     .     KSDELT( IS ) . LT . IDUMP ) THEN
C        IF( IJE5 . EQ . 8 ) THEN
       KSDELT( IS ) = IDUMP
       JEE( IS ) = 1
       NSS = NSS + 1
       JTRIG( NSS ) = IS
       END IF
  120  CONTINUE
C
       DO 130 IS = 1 , NSS
       JSE( IS ) = JTRIG( IS )
  130  CONTINUE
C
       MSS = NSS
```

```
             DO 140 KDIV = 1 , NOFDIV
             ITRIG = 0
             DO 150 KS = 1 , NSS
      C
             ISS = JSE( KS )
      C
             DO 160 KR = 1 , 3
             IVV = JS( KR , ISS )
      C
                IE = JV( 2 , IVV )
                IF( IE . GT . 0 ) THEN
      C
                IV1 = JE( 1 , IE )
                IF( IV1 . EQ . IVV ) THEN
                ISI = JE( 3 , IE )
                ELSE
                ISI = JE( 4 , IE )
                END IF
                IS = ISI
      C
      750       CONTINUE
      C
                JES = JEE( IS )
                XAS = XS( 3 ,  S )
                IF( JES . EQ . 0 . AND . XAS . LT . SAREVG ) THEN
                ITRIG = ITRIG + 1
                KTRIG( ITRIG ) = IS
                KSDELT( IS ) = IDUMP
                JEE( IS ) = 1
                END IF
      C
                DO 760 IR = 1 , 3
                JR = MOD( IR , 3 ) + 1
                IEA = IABS( JS( JR + 3 , IS ) )
                IF( IEA . EQ . IE ) THEN
                JJR = MOD( JR + 1 , 3 ) + 4
                IER = IABS( JS( JJR , IS ) )
      C
                IV1 = JE( 1 , IER )
                IF( IV1 . EQ . IVV ) THEN
                ISR = JE( 3 , IER )
                ELSE
                ISR = JE( 4 , IER )
                END IF
                END IF
      760       CONTINUE
      C
                IF( ISR . NE . ISI ) THEN
                IS = ISR
                IE = IER
      C
                GO TO 750
                END IF
      C
                ELSE
      C
                IE = - IE
                IV1 = JE( 1 , IE )
                IF( IV1 . EQ . IVV ) THEN
                ISI = JE( 3 , IE )
                ELSE
                ISI = JE( 4 , IE )
                END IF
                IS = ISI
                ISI = 0
      C
      650       CONTINUE
      C
                JES = JEE( IS )
                XAS = XS( 3 , IS )
                IF( JES . EQ . 0 . AND . XAS . LT . SAREVG ) THEN
                ITRIG = ITRIG + 1
                KTRIG( ITRIG ) = IS
                KSDELT( IS ) = IDUMP
```

```
                JEE( IS ) = 1
                END IF
      C
                DO 660 IR = 1 , 3
                JR = MOD( IR , 3 ) + 1
                IEA = IABS( JS( JR + 3 , IS ) )
                IF( IEA . EQ . IE ) THEN
                JJR = MOD( JR + 1 , 3 ) + 4
                IER = IABS( JS( JJR , IS ) )
      C
                IV1 = JE( 1 , IER )
                IF( IV1 . EQ . IVV ) THEN
                ISR = JE( 3 , IER )
                ELSE
                ISR = JE( 4 , IER )
                END IF
                END IF
      C
      660       CONTINUE
      C
                IF( ISR . NE . ISI ) THEN
                IS = ISR
                IE = IER
                GO TO 650
                END IF
                END IF
        160   CONTINUE
      C
        150   CONTINUE
      C
              DO 170 IS = 1 , ITRIG
              JTRIG( IS + MSS ) = KTRIG( IS )
              JSE( IS ) = KTRIG( IS )
        170   CONTINUE
              NSS = ITRIG
              MSS = MSS + ITRIG
      C
        140   CONTINUE
              NSS = MSS
      C
              DO 300 KDIV = 1 , 1
              LTRIG = NSS
              IEDGE = 0
              NCOLOR = 0
      C
              DO 290 IE = 1 , NE
              JSE( IE ) = 0
        290   CONTINUE
      C
              DO 310 IS = 1 , NSS
              ISS = JTRIG( IS )
              XSAREA = XS( 3 , ISS )
              IF( XSAREA . GE . RMINVG ) THEN
      C
              DO 335 IR = 4 , 6
              IE = IABS( JS( IR , ISS ) )
              IJE5 = JE( 5 , IE )
              IF( IJE5 . NE . 0 ) THEN
              JR2 = MOD( IR - 3 , 3 ) + 4
              IE2 = IABS( JS( JR2 , ISS ) )
              JR3 = MOD( IR - 2 , 3 ) + 4
              IE3 = IABS( JS( JR3 , ISS ) )
              XE1 = XE( 1 , IE )
              XE2 = XE( 1 , IE2 )
              XE3 = XE( 1 , IE3 )
              XEDIST = 1. / XE1
              YE2 = XE2 * XEDIST
              YE3 = XE3 * XEDIST
              ZE2 = ( YE2 - 1.5 ) * ( YE2 - .1 )
              ZE3 = ( YE3 - 1.5 ) * ( YE3 - .1 )
              YY2 = XE1 * XE1 + XE2 * XE2 + .35 * XE1 * XE2 - XE3 * XE3
              YY3 = XE1 * XE1 + XE3 * XE3 + .35 * XE1 * XE3 - XE2 * XE2
              IF( ZE2 . LT . .0 . AND . ZE3 . LT . 0. . AND .
           .      YY2 . GT . 0. . AND . YY3 . GT . 0. ) THEN
```

```
      CALL DISECT ( IE , IDONE , IDUMP )
C
      LTRIG = LTRIG + 1
      JTRIG( LTRIG ) = NS
      KSDELT( NS ) = IDUMP
C
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE
      JSE( NE ) = 1
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE - 1
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE - 1
      JSE( NE - 1 ) = 1
C
      END IF
      END IF
  335 CONTINUE
      END IF
  310 CONTINUE
C
      NSS = LTRIG
      IEDGE = 0
      NCOLOR = 0
C
      DO 295 IE = 1 , NE
      JSE( IE ) = 0
  295 CONTINUE
C
      DO 320 IS = 1 , NSS
      ISS = JTRIG( IS )
      XSAREA = XS( 3 , ISS )
C
      DO 735 IR = 4 , 6
      IE = IABS( JS( IR , ISS ) )
      IF( JSE( IE ) . EQ . 0 ) THEN
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = IE
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = IE
      JSE( IE ) = 1
      END IF
  735 CONTINUE
C
      IF( XSAREA . GT . RMINVG ) THEN
C
      AREAXS = SAREA( ISS )
      IE1 = IABS( JS( 4 , ISS ) )
      XE1 = XE( 1 , IE1 )
      HD1 = AREAXS * XE1 * XE1
      IJE5 = JE( 5 , IE1 )
      IE2 = IABS( JS( 5 , ISS ) )
      XE2 = XE( 1 , IE2 )
      HD2 = AREAXS * XE2 * XE2
      IJE5 = IJE5 + JE( 5 , IE2 )
      IE3 = IABS( JS( 6 , ISS ) )
      XE3 = XE( 1 , IE3 )
      HD3 = AREAXS * XE3 * XE3
      IJE5 = IJE5 + JE( 5 , IE3 )
      RATIO = AMAX1( HD1 , HD2 , HD3 )
      IRATIO = 0
      IF( RATIO . LE . 7. . AND . IJE5 . EQ . 0 . AND .
     .                      XSAREA . GT . SMINVG ) IRATIO = 1
      IF( IJE5 . GT . 0 ) IRATIO = 2
C
      IF( IRATIO . EQ . 2 ) THEN
      IJE51 = JE( 5 , IE1 )
      IJE52 = JE( 5 , IE2 )
      IJE53 = JE( 5 , IE3 )
      IF( IJE51 . NE . 0 ) THEN
      IEDIST = IE1
      XE1 = XE( 1 , IE1 )
```

```
            XE2 = XE( 1 , IE2 )
            XE3 = XE( 1 , IE3 )
            END IF
            IF( IJE52 . NE . 0 ) THEN
            IEDIST = IE2
            XE1 = XE( 1 , IE2 )
            XE2 = XE( 1 , IE1 )
            XE3 = XE( 1 , IE3 )
            END IF
            IF( IJE53 . NE . 0 ) THEN
            IEDIST = IE3
            XE1 = XE( 1 , IE3 )
            XE2 = XE( 1 , IE2 )
            XE3 = XE( 1 , IE1 )
            END IF
            XEDIST = 1. / XE( 1 , IEDIST )
            YE2 = XE2 * XEDIST
            YE3 = XE3 * XEDIST
            ZE2 = ( YE2 - 1.5 ) * ( YE2 - .1 )
            ZE3 = ( YE3 - 1.5 ) * ( YE3 - .1 )
            YY2 = XE1 * XE1 + XE2 * XE2 + .35 * XE1 * XE2 - XE3 * XE3
            YY3 = XE1 * XE1 + XE3 * XE3 + .35 * XE1 * XE3 - XE2 * XE2
            YY3 = XE1 * XE1 + XE3 * XE3 - XE2 * XE2
            IF( ZE2 . LT . .0 . AND . ZE3 . LT . 0. . AND .
     .          YY2 . GT . 0. . AND . YY3 . GT . 0. ) THEN
            CALL DISECT ( IEDIST , IDONE , IDUMP )
C
            LTRIG = LTRIG + 1
            JTRIG( LTRIG ) = NS
            KSDELT( NS ) = IDUMP
C
            IEDGE = IEDGE + 1
            IRECNC( IEDGE ) = NE
            NCOLOR = NCOLOR + 1
            JEE( NCOLOR ) = NE
            JSE( NE ) = 1
            IEDGE = IEDGE + 1
            IRECNC( IEDGE ) = NE - 1
            NCOLOR = NCOLOR + 1
            JEE( NCOLOR ) = NE - 1
            JSE( NE - 1 ) = 1
C
            END IF
            END IF
C
            IF( IRATIO . EQ . 1 ) THEN
C
            CALL VERCEN( ISS )
            KSDELT( ISS ) = IDUMP
            LTRIG = LTRIG + 1
            JTRIG( LTRIG ) = NS - 1
            KSDELT( NS - 1 ) = IDUMP
C
            LTRIG = LTRIG + 1
            JTRIG( LTRIG ) = NS
            KSDELT( NS ) = IDUMP
C
            IEDGE = IEDGE + 1
            IRECNC( IEDGE ) = NE
            NCOLOR = NCOLOR + 1
            JEE( NCOLOR ) = NE
            JSE( NE ) = 1
            IEDGE = IEDGE + 1
            IRECNC( IEDGE ) = NE - 1
            NCOLOR = NCOLOR + 1
            JEE( NCOLOR ) = NE - 1
            JSE( NE - 1 ) = 1
            IEDGE = IEDGE + 1
            IRECNC( IEDGE ) = NE - 2
            NCOLOR = NCOLOR + 1
            JEE( NCOLOR ) = NE - 2
            JSE( NE - 2 ) = 1
C
            ELSE
```

```
C
      IDISCT = 0
      DO 545 KK = 4 , 6
      IEE = JS( KK , ISS )
      IEF = IABS( IEE )
      IJE55 = JE( 5 , IEF )
      IF( IJE55 . EQ . 0 ) THEN
      IF( IEE . GT . 0 ) THEN
      ISI = JE( 4 , IEE )
      ELSE
      ISI = JE( 3 , IEF )
      END IF
      AREAXS = SAREA( ISI )
      IE1 = IABS( JS( 4 , ISI ) )
      XE1 = XE( 1 , IE1 )
      IJE55 = JE( 5 , IE1 )
      HD1 = AREAXS * XE1 * XE1
      IE2 = IABS( JS( 5 , ISI ) )
      XE2 = XE( 1 , IE2 )
      IJE55 = IJE55 + JE( 5 , IE2 )
      HD2 = AREAXS * XE2 * XE2
      IE3 = IABS( JS( 6 , ISI ) )
      XE3 = XE( 1 , IE3 )
      IJE55 = IJE55 + JE( 5 , IE3 )
      HD3 = AREAXS * XE3 * XE3
      RATIO = AMAX1( HD1 , HD2 , HD3 )
      YSAREA = XS( 3 , ISI )
      IF( RATIO . LT . 7. . AND . YSAREA . GT . SMINVG . AND .
     .                              IJE55 . EQ . 0 ) THEN
      IDISCT = 1
      DO 435 IR = 4 , 6
      IE = IABS( JS( IR , ISI ) )
      IF( JSE( IE ) . EQ . 0 ) THEN
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = IE
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = IE
      JSE( IE ) = 1
      END IF
435   CONTINUE
      CALL VERCEN( ISI )
      KSDELT( ISI ) = IDUMP
      LTRIG = LTRIG + 1
      JTRIG( LTRIG ) = NS - 1
      KSDELT( NS - 1 ) = IDUMP
C
      LTRIG = LTRIG + 1
      JTRIG( LTRIG ) = NS
      KSDELT( NS ) = IDUMP
C
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE
      JSE( NE ) = 1
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE - 1
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE - 1
      JSE( NE - 1 ) = 1
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE - 2
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE - 2
      JSE( NE - 2 ) = 1
      END IF
      END IF
545   CONTINUE
C
      IF( IDISCT . EQ . 0 ) THEN
      IE1 = IABS( JS( 4 , ISS ) )
      XE1 = XE( 1 , IE1 )
      IE2 = IABS( JS( 5 , ISS ) )
      XE2 = XE( 1 , IE2 )
```

```
      IE3 = IABS( JS( 6 , ISS ) )
      XE3 = XE( 1 , IE3 )
      IEDIST = IE1
      XEDIST = XE1
      IF( XE2 . GT . XEDIST ) THEN
      XEDIST = XE2
      IEDIST = IE2
      END IF
      IF( XE3 . GT . XEDIST ) THEN
      XEDIST = XE3
      IEDIST = IE3
      END IF
      ISL = JE( 3 , IEDIST )
      ISR = JE( 4 , IEDIST )
      XSISL = XS( 3 , ISL )
      XSISR = XS( 3 , ISR )
      IJE5 = JE( 5 , IEDIST )
      IF( XSISL . GT . RMINVG . AND . XSISR . GT . RMINVG . AND .
     .    IJE5 . EQ . 0 . AND . IRATIO . NE . 2 )  THEN
      IF( ISS . NE . ISL ) THEN
      DO 345 IR = 4 , 6
      IE = IABS( JS( IR , ISL ) )
      IF( JSE( IE ) . EQ . 0 ) THEN
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = IE
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = IE
      JSE( IE ) = 1
      END IF
  345 CONTINUE
      END IF
C
      IF( ISS . NE . ISR ) THEN
      DO 355 IR = 4 , 6
      IE = IABS( JS( IR , ISR ) )
      IF( JSE( IE ) . EQ . 0 ) THEN
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = IE
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = IE
      JSE( IE ) = 1
      END IF
  355 CONTINUE
      END IF
C
      IDONE = 0
      CALL DISECT ( IEDIST , IDONE , IDUMP )
      IF( IDONE . EQ . 1 ) THEN
C
      LTRIG = LTRIG + 1
      JTRIG( LTRIG ) = NS
      KSDELT( NS ) = IDUMP
      LTRIG = LTRIG + 1
      JTRIG( LTRIG ) = NS - 1
      KSDELT( NS - 1 ) = IDUMP
C
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE
      JSE( NE ) = 1
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE - 1
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE - 1
      JSE( NE - 1 ) = 1
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE - 2
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE - 2
      JSE( NE - 2 ) = 1
      END IF
C
      END IF
```

```
      END IF
      END IF
      END IF
C
  320 CONTINUE
C
      DO 340 IEM = 1 , NCOLOR
      IE = JEE( IEM )
C
      ISL = JE( 3 , IE )
      YSAREA = XS( 3 , ISL )
      IJE5 = JE( 5 , IE )
      IF( YSAREA . GE . RMINVG . AND . IJE5 . NE . 0 ) THEN
      IE1 = IABS( JS( 4 , ISL ) )
      IE2 = IABS( JS( 5 , ISL ) )
      IE3 = IABS( JS( 6 , ISL ) )
      IJE51 = JE( 5 , IE1 )
      IJE52 = JE( 5 , IE2 )
      IJE53 = JE( 5 , IE3 )
      IF( IJE51 . NE . 0 ) THEN
      IEDIST = IE1
      XE1 = XE( 1 , IE1 )
      XE2 = XE( 1 , IE2 )
      XE3 = XE( 1 , IE3 )
      END IF
      IF( IJE52 . NE . 0 ) THEN
      IEDIST = IE2
      XE1 = XE( 1 , IE2 )
      XE2 = XE( 1 , IE1 )
      XE3 = XE( 1 , IE3 )
      END IF
      IF( IJE53 . NE . 0 ) THEN
      IEDIST = IE3
      XE1 = XE( 1 , IE3 )
      XE2 = XE( 1 , IE2 )
      XE3 = XE( 1 , IE1 )
      END IF
      XEDIST = 1. / XE( 1 , IEDIST )
      YE2 = XE2 * XEDIST
      YE3 = XE3 * XEDIST
      ZE2 = ( YE2 - 1.5 ) * ( YE2 - .1 )
      ZE3 = ( YE3 - 1.5 ) * ( YE3 - .1 )
      YY2 = XE1 * XE1 + XE2 * XE2 + .35 * XE1 * XE2 - XE3 * XE3
      YY3 = XE1 * XE1 + XE3 * XE3 + .35 * XE1 * XE3 - XE2 * XE2
      IF( ZE2 . LT . .0 . AND . ZE3 . LT . 0. . AND .
     .    YY2 . GT . 0. . AND . YY3 . GT . 0. ) THEN
      CALL DISECT ( IEDIST , IDONE , IDUMP )
C
      LTRIG = LTRIG + 1
      JTRIG( LTRIG ) = NS
      KSDELT( NS ) = IDUMP
C
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE
      JSE( NE ) = 1
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE - 1
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE - 1
      JSE( NE - 1 ) = 1
C
       ELSE
C
      IEDIST = IE1
      XEDIST = XE1
      IF( XE2 . GT . XEDIST ) THEN
      XEDIST = XE2
      IEDIST = IE2
      END IF
      IF( XE3 . GT . XEDIST ) THEN
      XEDIST = XE3
      IEDIST = IE3
```

```
      END IF
      ISL = JE( 3 , IEDIST )
      ISR = JE( 4 , IEDIST )
      XSISL = XS( 3 , ISL )
      XSISR = XS( 3 , ISR )
      IJE5 = JE( 5 , IEDIST )
      IF( XSISL . GT . RMINVG . AND . XSISR . GT . RMINVG . AND .
     .    IJE5 . EQ . 0 )  THEN
      DO 645 IR = 4 , 6
      IE = IABS( JS( IR , ISL ) )
      IF( JSE( IE ) . EQ . 0 ) THEN
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = IE
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = IE
      JSE( IE ) = 1
      END IF
  645 CONTINUE
      DO 655 IR = 4 , 6
      IE = IABS( JS( IR , ISR ) )
      IF( JSE( IE ) . EQ . 0 ) THEN
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = IE
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = IE
      JSE( IE ) = 1
      END IF
  655 CONTINUE
C
      IDONE = 0
      CALL DISECT ( IEDIST , IDONE , IDUMP )
      IF( IDONE . EQ . 1 ) THEN
C
      LTRIG = LTRIG + 1
      JTRIG( LTRIG ) = NS
      KSDELT( NS ) = IDUMP
      LTRIG = LTRIG + 1
      JTRIG( LTRIG ) = NS - 1
      KSDELT( NS - 1 ) = IDUMP
C
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE
      JSE( NE ) = 1
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE - 1
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE - 1
      JSE( NE - 1 ) = 1
      IEDGE = IEDGE + 1
      IRECNC( IEDGE ) = NE - 2
      NCOLOR = NCOLOR + 1
      JEE( NCOLOR ) = NE - 2
      JSE( NE - 2 ) = 1
C
      END IF
      END IF
      END IF
      END IF
  340 CONTINUE
C
      NSS = LTRIG
C
      DO 370 IEM = 1 , NCOLOR
      IE = JEE( IEM )
      CALL RECNC( IE , IDONE , ITL , ITR , JA , JB , JC , JD )
      CALL RECNC( JA , JADONE , ITL , ITR , JAA , JAB , JAC , JAD )
      CALL RECNC( JB , JBDONE , ITL , ITR , JBA , JBB , JBC , JBD )
      CALL RECNC( JC , JCDONE , ITL , ITR , JCA , JCB , JCC , JCD )
      CALL RECNC( JD , JDDONE , ITL , ITR , JDA , JDB , JDC , JDD )
  370 CONTINUE
C
  300 CONTINUE
```

```
C
      NVECE = NE / MBL
      NREME = NE - NVECE * MBL
      NVECS = NS / MBL
      NREMS = NS - NVECS * MBL
      NVECV = NV / MBL
      NREMV = NV - NVECV * MBL
C
      DO 400 INE = 1 , NVECE
      NOFVEE( INE ) = MBL
  400 CONTINUE
      NVEEE = NVECE
      IF( NREME . GT . 0 ) THEN
      NVEEE = NVECE + 1
      NOFVEE( NVEEE ) = NREME
      END IF
C
      DO 410 INS = 1 , NVECS
      NOFVES( INS ) = MBL
  410 CONTINUE
      NVEES = NVECS
      IF( NREMS . GT . 0 ) THEN
      NVEES = NVECS + 1
      NOFVES( NVEES ) = NREMS
      END IF
C
      DO 420 INV = 1 , NVECV
      NOFVEV( INV ) = MBL
  420 CONTINUE
      NVEEV = NVECV
      IF( NREMV . GT . 0 ) THEN
      NVEEV = NVECV + 1
      NOFVEV( NVEEV ) = NREMV
      END IF
C
       PRINT*,NV,NE,NS
C
C --- EXIT POINT FROM SUBROUTINE ---------------------------------------
C
C     ------
      RETURN
C     ------
C
C     ---
      END
```

```
      SUBROUTINE DELPTNT( DAREA , IDUMP )
C
C----------------------------------------------------------------I
C                                                                I
C       DELPTN ADAPT THE GRID DYNAMICALLY, DELETE VERTECES       I
C       WILL FLAGED TRIANGLES FOR DELETION                       I
C                                                                I
C----------------------------------------------------------------I
C
          IMPLICIT REAL (A-H,O-Z)
C
      include     'cmsh00.h'
      include     'chyd00.h'
      include     'cint00.h'
      include     'cphs10.h'
      include     'cphs20.h'
C
      INTEGER JTRIG(MEM),KTRIG(MEM),IRECNC(MEM)
      INTEGER JSE(MEM),JEE(MEM),IOFDVS(10),NOFDVS(10)
      INTEGER IITRIG(200)
      REAL ADFCTR(8),DLFCTR(8)
C
      EQUIVALENCE (UL,JTRIG)
      EQUIVALENCE (VR,KTRIG)
      EQUIVALENCE (VL,IRECNC)
      EQUIVALENCE (PR,JSE)
      EQUIVALENCE (PL,JEE)
C
       DLFCTR( 1 ) = DAREA
       DLFCTR( 2 ) = .4
       DLFCTR( 3 ) = .5
       DLFCTR( 4 ) = .65
       DLFCTR( 5 ) = .8
C
       SMINVG = SAREVG * DAREA
       DO 112 IS = 1 , NS
       JSDELT( IS  ) = 0
  112  CONTINUE
       ISDELT = 0
C
       NSS = 0
       FLUXPP = .00001 * HYDMOM( 4 )
       FLUXUU = .00001 * HYDMOM( 2 )
       FLUXRR = .00001 * HYDMOM( 1 )
       DO 120 IS = 1 , NS
       PCRTRY = HYDFLX( IS , 4 ) - FLUXPP
       IPCRTR = SIGN( 1. , PCRTRY )
       UCRTRY = HYDFLX( IS , 2 ) - FLUXUU
       IUCRTR = SIGN( 1. , UCRTRY )
       RCRTRY = HYDFLX( IS , 1 ) - FLUXRR
       IRCRTR = SIGN( 1. , RCRTRY )
       NIDUMP = IDUMP - NAREAD
       IF(
     .     IPCRTR . EQ . - 1 . AND .
     .     IUCRTR . EQ . - 1 . AND .
     .     IRCRTR . EQ . - 1 . AND .
     .    KSDELT( IS ) . LE . NIDUMP . AND .
     .    KSDELT( IS ) . NE . 0 ) THEN
       NSS = NSS + 1
       JTRIG( NSS ) = IS
       END IF
  120  CONTINUE
C
       PRINT*,NV,NE,NS,NSS
C
       ISDELT = NSS
       DO 210 IS = 1 , NSS
       JSDELT( IS ) = JTRIG( IS )
  210  CONTINUE
C
       DO 300 KDIV = 1 , 1
       ILOOP = 1
  310  CONTINUE
       ISS = JSDELT( ILOOP )
```

```
        XSAREA = XS( 3 , ISS )
        INDCTR = 0
        IF( XSAREA . LT . SMINVG ) THEN
C
        CALL VERDELT( ISS , INDCTR , NIDUMP , JJTRIG , IITRIG )
        IF( INDCTR . EQ . 1 ) THEN
        ILOOP = 1
        ELSE
        JSDELT( ILOOP ) = 0
        ILOOP = ILOOP + 1
        END IF
        ELSE
        JSDELT( ILOOP ) = 0
        ILOOP = ILOOP + 1
        END IF
C
        IF( ISDELT . GT . ILOOP ) GO TO 310
        PRINT *,KDIV,NV,NE,NS,DLFCTR(KDIV)
  300   CONTINUE
C
      NVECE = NE / MBL
      NREME = NE - NVECE * MBL
      NVECS = NS / MBL
      NREMS = NS - NVECS * MBL
      NVECV = NV / MBL
      NREMV = NV - NVECV * MBL
C
      DO 400 INE = 1 , NVECE
      NOFVEE( INE ) = MBL
  400 CONTINUE
      NVEEE = NVECE
      IF( NREME . GT . 0 ) THEN
      NVEEE = NVECE + 1
      NOFVEE( NVEEE ) = NREME
      END IF
C
      DO 410 INS = 1 , NVECS
      NOFVES( INS ) = MBL
  410 CONTINUE
      NVEES = NVECS
      IF( NREMS . GT . 0 ) THEN
      NVEES = NVECS + 1
      NOFVES( NVEES ) = NREMS
      END IF
C
      DO 420 INV = 1 , NVECV
      NOFVEV( INV ) = MBL
  420 CONTINUE
      NVEEV = NVECV
      IF( NREMV . GT . 0 ) THEN
      NVEEV = NVECV + 1
      NOFVEV( NVEEV ) = NREMV
      END IF
C
       PRINT*,NV,NE,NS
C
C --- EXIT POINT FROM SUBROUTINE -------------------------------------
C
C     ------
      RETURN
C     ------
C
C     ---
      END
```

```
      SUBROUTINE RELAXY( IV )
      IMPLICIT REAL (A-H,O-Z)
C
C-----------------------------------------------------------I
C                                                           I
C     THIS ROUTINE RELAX THE GRID AFTER DELETION            |
C                                                           I
C-----------------------------------------------------------I
C
      include     'cmsh00.h'
      include     'chyd00.h'
      include     'cint00.h'
      include     'cphs10.h'
      include     'cphs20.h'
         ITRIG = 0
         IETRIG = 0
         IE = JV( 2 , IV )
         IF( IE . GT . 0 ) THEN
C
         IV1 = JE( 1 , IE )
         IV2 = JE( 2 , IE )
         IF( IV1 . EQ . IV ) THEN
         ISI = JE( 3 , IE )
         ELSE
         ISI = JE( 4 , IE )
         END IF
         IS = ISI
C
 75      CONTINUE
C
         DO 65 IR = 1 , 3
         JR = MOD( IR , 3 ) + 1
         IEA = IABS( JS( JR + 3 , IS ) )
         IF( IEA . EQ . IE ) THEN
         IIR = MOD( JR , 3 ) + 4
         IEI = JS( IIR , IS )
         IEII = IABS( IEI )
         IETRIG = IETRIG + 1
         JECRSS( IETRIG ) = IEII
         JJR = MOD( JR + 1 , 3 ) + 4
         IEM = JS( JJR , IS )
         IER = IABS( IEM )
         IETRIG = IETRIG + 1
         JECRSS( IETRIG ) = IER
C
         IV1 = JE( 1 , IER )
         IV2 = JE( 2 , IER )
         IF( IV1 . EQ . IV ) THEN
         ISR = JE( 3 , IER )
         ITRIG = ITRIG + 1
         IICOLR( ITRIG ) = IV2
         JSCRSS( ITRIG ) = ISR
         ELSE
         ISR = JE( 4 , IER )
         ITRIG = ITRIG + 1
         IICOLR( ITRIG ) = IV1
         JSCRSS( ITRIG ) = ISR
         END IF
         END IF
 65      CONTINUE
C
         IF( ISR . NE . ISI ) THEN
         IS = ISR
         IE = IER
         GO TO 75
         END IF
C
         DO 510 IE = 1 , ITRIG
C
         IEM = MOD( IE - 1 , ITRIG ) + 1
         IEP = MOD( IE , ITRIG ) + 1
         IEI = MOD( IE + 1 , ITRIG ) + 1
C
         IV1 = IICOLR( IEM )
```

```
      IV2 = IICOLR( IEP )
      IV3 = IICOLR( IEI )
C
      X1 = XV( 1 , IV1 ) - XV( 1 , IV2 )
      Y1 = XV( 2 , IV1 ) - XV( 2 , IV2 )
      X2 = XV( 1 , IV3 ) - XV( 1 , IV2 )
      Y2 = XV( 2 , IV3 ) - XV( 2 , IV2 )
      XSIN = ( X2 * Y1 - X1 * Y2 )
      XCOS = ( X1 * X2 + Y1 * Y2 )
      ANGLE( IE ) = XSIN / ( ABS( XCOS ) + 1.E-7 )
      IF( ANGLE( IE ) . LT . 0. ) RETURN
C
 510  CONTINUE
C
      XSUM = 0.
      YSUM = 0.
      HSUMR = 0.
      HSUMU = 0.
      HSUMV = 0.
      HSUMP = 0.
      HSUMG = 0.
C
      DO 110 IT = 1 , ITRIG
      IVV = IICOLR( IT )
C
      XSUM = XSUM + XV( 1 , IVV )
      YSUM = YSUM + XV( 2 , IVV )
C
      HSUMR  = HSUMR + HYDVVV( IVV , 1 )
      HSUMU  = HSUMU + HYDVVV( IVV , 2 )
      HSUMV  = HSUMV + HYDVVV( IVV , 3 )
      HSUMP  = HSUMP + HYDVVV( IVV , 4 )
      HSUMG  = HSUMG + HYDVVV( IVV , 5 )
  110 CONTINUE
C
      XINVRG = 1. / ITRIG
      XV( 1 , IV ) = XSUM * XINVRG
      XV( 2 , IV ) = YSUM * XINVRG
      HYDVVV( IV , 1 ) = HSUMR * XINVRG
      HYDVVV( IV , 2 ) = HSUMU * XINVRG
      HYDVVV( IV , 3 ) = HSUMV * XINVRG
      HYDVVV( IV , 4 ) = HSUMP * XINVRG
      HYDVVV( IV , 5 ) = HSUMG * XINVRG
C
      ELSE
C
      IE = - IE
      IV1 = JE( 1 , IE )
      IV2 = JE( 2 , IE )
      IF( IV1 . EQ . IV ) THEN
      ISI = JE( 3 , IE )
      ITRIG = ITRIG + 1
      JSCRSS( ITRIG ) = ISI
      IICOLR( ITRIG ) = IV2
      ELSE
      ISI = JE( 4 , IE )
      ITRIG = ITRIG + 1
      JSCRSS( ITRIG ) = ISI
      IICOLR( ITRIG ) = IV1
      END IF
C
      IS = ISI
      ISI = 0
      IIE = IE
      IETRIG = IETRIG + 1
      JECRSS( IETRIG ) = IE
C
 670  CONTINUE
C
      DO 680 IR = 1 , 3
      JR = MOD( IR , 3 ) + 1
      IEA = IABS( JS( JR + 3 , IS ) )
      IF( IEA . EQ . IE ) THEN
      IIR = MOD( JR , 3 ) + 4
```

```
            IEI = JS( IIR , IS )
            IEII = IABS( IEI )
            IETRIG = IETRIG + 1
            JECRSS( IETRIG ) = IEII
            JJR = MOD( JR + 1 , 3 ) + 4
            IEM = JS( JJR , IS )
            IER = IABS( IEM )
            IETRIG = IETRIG + 1
            JECRSS( IETRIG ) = IER
C
            IV1 = JE( 1 , IER )
            IV2 = JE( 2 , IER )
            IF( IV1 . EQ . IV ) THEN
            ISR = JE( 3 , IER )
            ITRIG = ITRIG + 1
            IICOLR( ITRIG ) = IV2
            JSCRSS( ITRIG ) = ISR
            ELSE
            ISR = JE( 4 , IER )
            ITRIG = ITRIG + 1
            IICOLR( ITRIG ) = IV1
            JSCRSS( ITRIG ) = ISR
            END IF
            END IF
C
 680        CONTINUE
C
            IF( ISR . NE . ISI ) THEN
            IS = ISR
            IE = IER
            GO TO 670
            END IF
            ITRIG = ITRIG - 1
C
            IV1 = JE( 1 , IIE )
            IV2 = JE( 2 , IIE )
C
            IV3 = JE( 1 , IER )
            IV4 = JE( 2 , IER )
C
            X1 = XV( 1 , IV1 ) - XV( 1 , IV2 )
            Y1 = XV( 2 , IV1 ) - XV( 2 , IV2 )
            X2 = XV( 1 , IV4 ) - XV( 1 , IV3 )
            Y2 = XV( 2 , IV4 ) - XV( 2 , IV3 )
            XSIN = ( X2 * Y1 - X1 * Y2 )
            XCOS = ( X1 * X2 + Y1 * Y2 )
            XANGLE = XSIN / ( ABS( XCOS ) + 1.E-7 )
C
            IF( ABS( XANGLE ) . GT . 1.E-3 ) RETURN
C
            IVI = IV1
            IF( IV . EQ . IV1 ) IVI = IV2
            IVL = IV3
            IF( IV . EQ . IV3 ) IVL = IV4
            IVTRIG = ITRIG + 1
            IICOLR( IVTRIG ) = IVL
C
            DO 512 IE = 1 , IVTRIG
C
            IEM = MOD( IE - 1 , IVTRIG ) + 1
            IEP = MOD( IE , IVTRIG ) + 1
            IEI = MOD( IE + 1 , IVTRIG ) + 1
C
            IV1 = IICOLR( IEM )
            IV2 = IICOLR( IEP )
            IV3 = IICOLR( IEI )
C
            X1 = XV( 1 , IV1 ) - XV( 1 , IV2 )
            Y1 = XV( 2 , IV1 ) - XV( 2 , IV2 )
            X2 = XV( 1 , IV3 ) - XV( 1 , IV2 )
            Y2 = XV( 2 , IV3 ) - XV( 2 , IV2 )
            XSIN = ( X2 * Y1 - X1 * Y2 )
            XCOS = ( X1 * X2 + Y1 * Y2 )
            ANGLE( IE ) = XSIN / ( ABS( XCOS ) + 1.E-7 )
```

```
            IF( ANGLE( IE ) . LT . 0. ) RETURN
C
  512       CONTINUE
C
            XV( 1 , IV ) = .5 * ( XV( 1 , IVI ) + XV( 1 , IVL ) )
            XV( 2 , IV ) = .5 * ( XV( 2 , IVI ) + XV( 2 , IVL ) )
            HYDVVV( IV , 1 ) = .5 * ( HYDVVV( IVI , 1 ) +
     .                                HYDVVV( IVL , 1 ) )
            HYDVVV( IV , 2 ) = .5 * ( HYDVVV( IVI , 2 ) +
     .                                HYDVVV( IVL , 2 ) )
            HYDVVV( IV , 3 ) = .5 * ( HYDVVV( IVI , 3 ) +
     .                                HYDVVV( IVL , 3 ) )
            HYDVVV( IV , 4 ) = .5 * ( HYDVVV( IVI , 4 ) +
     .                                HYDVVV( IVL , 4 ) )
            HYDVVV( IV , 5 ) = .5 * ( HYDVVV( IVI , 5 ) +
     .                                HYDVVV( IVL , 5 ) )
C
            END IF
C
            DO 120 ISNN = 1 , ITRIG
            INS = JSCRSS( ISNN )
C
            IV1 = JS( 1 , INS )
            IV2 = JS( 2 , INS )
            IV3 = JS( 3 , INS )
            AX = XV( 1 , IV2 ) - XV( 1 , IV1 )
            AY = XV( 2 , IV2 ) - XV( 2 , IV1 )
            BX = XV( 1 , IV3 ) - XV( 1 , IV1 )
            BY = XV( 2 , IV3 ) - XV( 2 , IV1 )
            XS( 3 , INS ) = 0.5 * ( AX * BY - AY * BX )
C
            SAREA( INS ) = 1. / XS( 3 , INS )
            HYDFLX( INS , 4 ) = 0.
            HYDFLX( INS , 1 ) = 0.
            HYDFLX( INS , 2 ) = 0.
            KSDELT( INS ) = 1
C
            XXC = ( XV( 1 , IV1 ) + XV( 1 , IV2 ) + XV( 1 , IV3 ) ) *
     .            THIRD
            YYC = ( XV( 2 , IV1 ) + XV( 2 , IV2 ) + XV( 2 , IV3 ) ) *
     .            THIRD
            XS( 1 , INS ) = XXC
            XS( 2 , INS ) = YYC
C
            DO 130 IR = 1 , MHQ
            HYDV( INS , IR ) = ( HYDVVV( IV1 , IR ) +
     .                           HYDVVV( IV2 , IR ) +
     .                           HYDVVV( IV3 , IR ) ) * THIRD
  130       CONTINUE
C
            HDUM          = 1. / ( HYDV( INS , 1 ) + 1.E-12 )
            HYDV( INS , 2 ) = HYDV( INS , 2 ) * HDUM
            HYDV( INS , 3 ) = HYDV( INS , 3 ) * HDUM
            HYDV( INS , 4 ) = ( HYDV( INS , 4 )  -
     .                    .5 * HYDV( INS , 1 ) *
     .      ( HYDV( INS , 2 ) * HYDV( INS , 2 ) +
     .        HYDV( INS , 3 ) * HYDV( INS , 3 ) ) ) *
     .                      ( HYDV( INS , 5 ) - 1. )
C
  120       CONTINUE
C
            DO 140 IENN = 1 , IETRIG
            IEN = JECRSS( IENN )
C
            JV1 = JE( 1 , IEN )
            JV2 = JE( 2 , IEN )
            AX = XV( 1 , JV2 ) - XV( 1 , JV1 )
            AY = XV( 2 , JV2 ) - XV( 2 , JV1 )
            XE( 1 , IEN ) = SQRT( AX * AX + AY * AY )
            XEREV = 1. / XE( 1 , IEN )
            XN( IEN ) = AY * XEREV
            YN( IEN ) = - AX * XEREV
            ISSR = JE( 4 , IEN )
            ISSL = JE( 3 , IEN )
```

```
C
            IF( JE( 5 , IEN ) . NE . 0 ) THEN
C
            AA = XV( 1 , JV2 ) - XV( 1 , JV1 )
            BB = XV( 2 , JV2 ) - XV( 2 , JV1 )
            XEL = XS( 1 , ISSL )
            YEL = XS( 2 , ISSL )
            CC = XEL - XV( 1 , JV1 )
            DD = YEL - XV( 2 , JV1 )
            EE = ( AA * CC + BB * DD ) * XEREV * XEREV
            XER = XV( 1 , JV1 ) + AA * EE
            YER = XV( 2 , JV1 ) + BB * EE
            AX = XER - XEL
            AY = YER - YEL
            XE( 2 , IEN ) = SQRT( AX * AX + AY * AY )
            XEREV = 1. / XE( 2 , IEN )
            XXN( IEN ) = AX * XEREV
            YYN( IEN ) = AY * XEREV
            XE( 2 , IEN ) = 2. * XE( 2 , IEN )
            XYMIDL( IEN ) = .5
            XMIDL( IEN ) = XER
            YMIDL( IEN ) = YER
C
            ELSE
C
            XER = XS( 1 , ISSR )
            YER = XS( 2 , ISSR )
            XEL = XS( 1 , ISSL )
            YEL = XS( 2 , ISSL )
C
            AA = XV( 1 , JV2 ) - XV( 1 , JV1 )
            BB = XV( 2 , JV2 ) - XV( 2 , JV1 )
            CC = XEL - XER
            DD = YEL - YER
            ACA = XER - XV( 1 , JV1 )
            DBD = YER - XV( 2 , JV1 )
            EE = ( ACA * DD - DBD * CC ) / ( AA * DD - BB * CC )
            XMIDL( IEN ) = XV( 1 , JV1 ) + AA * EE
            YMIDL( IEN ) = XV( 2 , JV1 ) + BB * EE
C
            XEMID = XMIDL( IEN ) - XEL
            YEMID = YMIDL( IEN ) - YEL
C
            AX = XER - XEL
            AY = YER - YEL
            XE( 2 , IEN ) = SQRT( AX * AX + AY * AY )
            XEREV = 1. / XE( 2 , IEN )
            XXN( IEN ) = AX * XEREV
            YYN( IEN ) = AY * XEREV
C
            XYMIDL( IEN ) = SQRT( XEMID * XEMID + YEMID * YEMID ) * XEREV
C
            END IF
C
140         CONTINUE
C
            DO 142 IENN = 1 , IETRIG
            IE = JECRSS( IENN )
            CALL RECNC( IE , IDONE , ITL , ITR , JA , JB , JC , JD )
            CALL RECNC( JA , JADONE , ITL , ITR , JAA , JAB , JAC , JAD )
            CALL RECNC( JB , JBDONE , ITL , ITR , JBA , JBB , JBC , JBD )
            CALL RECNC( JC , JCDONE , ITL , ITR , JCA , JCB , JCC , JCD )
            CALL RECNC( JD , JDDONE , ITL , ITR , JDA , JDB , JDC , JDD )
142         CONTINUE
C
C --- EXIT POINT FROM SUBROUTINE ---------------------------------------
C
C     ------
      RETURN
C     ------
C
C     ---
      END
```

```
      SUBROUTINE LAPLAC
C
C--------------------------------------------------------------I
C                                                              I
C          LAPLAC COMPUTE THE LAPLACIAN FOR GRID ADAPTATION    I
C                                                              I
C--------------------------------------------------------------I
C
      include     'cmsh00.h'
      include     'chyd00.h'
      include     'cint00.h'
      include     'cphs10.h'
      include     'cphs20.h'
C
      REAL RRMIDL(MBP),PPMIDL(MBP)
      REAL ROR(3),UOR(3),VOR(3),POR(3)
      REAL ROL(3),UOL(3),VOL(3),POL(3)
C
      EPSLON = .025
C
      DO 120 IS = 1 , NS
      RR( IS ) = 0.
      RL( IS ) = 0.
  120 CONTINUE
C
C --- BEGIN LOOP OVER ALL EDGES IN THE DOMAIN ----------------------
C
      NE1 = 1
      NE2 = NOFVEE( 1 )
      DO 90 INE = 1 , NVEEE
C
C --- FETCH HYDRO QUANTITIES ---------------------------------------
C
      DO 105 IE = NE1 , NE2
          KE = IE - NE1 + 1
C
       ISL = JE( 3 , IE )
       ISR = JE( 4 , IE )
C
      IF( JE ( 5 , IE ) . EQ . 0 ) THEN
C
         RRMDL = XYMIDL( IE ) * ( RGRAD( ISR , 1 ) -
     .                RGRAD( ISL , 1 ) ) + RGRAD( ISL , 1 )
         RLMDL = XYMIDL( IE ) * ( RGRAD( ISR , 2 ) -
     .                RGRAD( ISL , 2 ) ) + RGRAD( ISL , 2 )
         PRMDL = XYMIDL( IE ) * ( PGRAD( ISR , 1 ) -
     .                PGRAD( ISL , 1 ) ) + PGRAD( ISL , 1 )
         PLMDL = XYMIDL( IE ) * ( PGRAD( ISR , 2 ) -
     .                PGRAD( ISL , 2 ) ) + PGRAD( ISL , 2 )
C
      ELSE
C
         RRMDL = RGRAD( ISL , 1 )
         RLMDL = RGRAD( ISL , 2 )
         PRMDL = PGRAD( ISL , 1 )
         PLMDL = PGRAD( ISL , 2 )
C
      END IF
C
         RRMIDL( KE ) = ( RRMDL * XN( IE ) + RLMDL * YN( IE ) ) *
     .                                            XE( 1 , IE )
         PPMIDL( KE ) = ( PRMDL * XN( IE ) + PLMDL * YN( IE ) ) *
     .                                            XE( 1 , IE )
C
  105 CONTINUE
C
      DO 130 IE = NE1 , NE2
          KE = IE - NE1 + 1
C
       ISL = JE( 3 , IE )
       ISR = JE( 4 , IE )
C
      IF( JE( 5 , IE ) . EQ . 0 ) THEN
C
```

```
            RR( ISL ) = RR( ISL ) + RRMIDL( KE )
            RR( ISR ) = RR( ISR ) - RRMIDL( KE )
            RL( ISL ) = RL( ISL ) + PPMIDL( KE )
            RL( ISR ) = RL( ISR ) - PPMIDL( KE )
C
            ELSE
C
            RR( ISL ) = RR( ISL ) + RRMIDL( KE )
            RL( ISL ) = RL( ISL ) + PPMIDL( KE )
C
            END IF
C
  130       CONTINUE
C
            NE1 = NE2 + 1
            NE2 = NE2 + NOFVEE( INE + 1 )
   90       CONTINUE
C
            DO 135 IS = 1 , NS
            ZRR = ABS( RR( IS ) ) * SAREA( IS )
            ZPR = ABS( RL( IS ) ) * SAREA( IS )
            RR( IS ) = ZRR * SAREVG
            RL( IS ) = ZPR * SAREVG
  135       CONTINUE
C
            DO 140 IS = 1 , NS
            ZRL = ( RGRAD( IS , 1 ) * RGRAD( IS , 1 ) +
     .              RGRAD( IS , 2 ) * RGRAD( IS , 2 ) ) * SAREVG
            ZPL = ( PGRAD( IS , 1 ) * PGRAD( IS , 1 ) +
     .              PGRAD( IS , 2 ) * PGRAD( IS , 2 ) ) * SAREVG
            ZRR = ABS( HYDV( IS , 1 ) ) * EPSLON
            ZPP = ABS( HYDV( IS , 4 ) ) * EPSLON
            RR( IS ) = RR( IS ) / ( ZRL + ZRR )
            RL( IS ) = RL( IS ) / ( ZPL + ZPP )
  140       CONTINUE
C
C --- EXIT POINT FROM SUBROUTINE -------------------------------------
C
C           ------
            RETURN
C           ------
C
C           ---
            END
```

```
      SUBROUTINE RECNC( IE , IDONE , ITL , ITR , JA , JB , JC , JD )
      IMPLICIT REAL (A-H,O-Z)
C
C-------------------------------------------------------------I
C                                                             I
C     THIS ROUTINE CHECKS FOR RECONNECTION OF EDGE NUMBER IE  I
C     TO GET A BETTER CONNECTIVITY BETWEEN ADJACENT TRIANGLES I
C     USED AFTER ADDITION AND DELETION                        I
C                                                             I
C-------------------------------------------------------------I
C
      include     'cmsh00.h'
      include     'chyd00.h'
      include     'cint00.h'
      include     'cphs10.h'
      include     'cphs20.h'
C
          EROR = 1.0E-3
C
          IDONE = 0
          IF( IE . EQ . 0 ) RETURN
          IF( JE( 5 , IE ) . NE . 0 ) RETURN
          ITR = JE( 4 , IE )
          ITL = JE( 3 , IE )
C
C     IDENTIFY VERTICES
C
          I1 = JE( 1 , IE )
          I2 = JE( 2 , IE )
          DO 1 IV = 1 , 3
          ID = JS( IV , ITL )
          IF( ID . NE . I1 . AND . ID . NE . I2 ) THEN
          I4 = ID
          IV4 = IV
          END IF
1         CONTINUE
C
          DO 3 IV = 1 , 3
          ID = JS( IV , ITR )
          IF( ID . NE . I1 . AND . ID . NE . I2 ) THEN
          I3 = ID
          IV3 = IV
          END IF
3         CONTINUE
C     IT MAY HAPPEN THAT I3 IS I4.
          IF( I3 . EQ . I4 ) GO TO 999
C
C     COMPARE OPPOSING ANGLE PAIRS IN THE QUADRILATERAL AND RECONNECT TO
C     PRESERVE DIAGONAL DOMINANCE OF THE POISSON SOLVER.
C
          AX = XV( 1 , I3 ) - XV( 1 , I1 )
          AY = XV( 2 , I3 ) - XV( 2 , I1 )
          BX = XV( 1 , I4 ) - XV( 1 , I1 )
          BY = XV( 2 , I4 ) - XV( 2 , I1 )
          CX = XV( 1 , I4 ) - XV( 1 , I2 )
          CY = XV( 2 , I4 ) - XV( 2 , I2 )
          DX = XV( 1 , I3 ) - XV( 1 , I2 )
          DY = XV( 2 , I3 ) - XV( 2 , I2 )
          AI2 = AX * BY - AY * BX
          AI1 = CX * DY - CY * DX
          XLN = XE( 1 , IE )
          ROUNDF = EROR * XLN * XLN
C
C     IA IS BETWEEN I1 AND I3
C     IB IS BETWEEN I1 AND I4
C     IC IS BETWEEN I2 AND I4
C     ID IS BETWEEN I2 AND I3
C
          IB = JS( IV4 + 3 , ITL )
          ID = JS( IV3 + 3 , ITR )
          IV4 = MOD( IV4 + 1 , 3 ) + 1
          IV3 = MOD( IV3 + 1 , 3 ) + 1
          IC = JS( IV4 + 3 , ITL )
          IA = JS( IV3 + 3 , ITR )
```

```
C
      JB = IABS( IB )
      JD = IABS( ID )
      JA = IABS( IA )
      JC = IABS( IC )
      IF( AI2 . LT . ROUNDF . OR . AI1 . LT . ROUNDF ) RETURN
C
      XL1 = XE( 1 , JA )
      XL2 = XE( 1 , JB )
      XL3 = XE( 1 , JC )
      XL4 = XE( 1 , JD )
C
      XX = XV( 1 , I3 ) - XV( 1 , I4 )
      YY = XV( 2 , I3 ) - XV( 2 , I4 )
      XLL = SQRT( XX * XX + YY * YY )
C
      AREATL = SAREA( ITL )
      AREATR = SAREA( ITR )
      ASP2 = AREATL * XL2 * XL2
      ASP3 = AREATL * XL3 * XL3
      ASPTL = AREATL * XLN * XLN
      ASP1 = AREATR * XL1 * XL1
      ASP4 = AREATR * XL4 * XL4
      ASPTR = AREATR * XLN * XLN
      ASPN = AMAX1( ASPTL , ASPTR , ASP1 , ASP2 , ASP3 , ASP4 )
C
      XSISR = 0.5 * AI2
      XSINSR = 1. / XSISR
C
      XSISL = 0.5 * AI1
      XSINSL = 1. / XSISL
C
      ASP2 = XSINSR * XL2 * XL2
      ASP1 = XSINSR * XL1 * XL1
      ASPSR = XSINSR * XLL * XLL
      ASP3 = XSINSL * XL3 * XL3
      ASP4 = XSINSL * XL4 * XL4
      ASPSL = XSINSL * XLL * XLL
      ASPL = AMAX1( ASPSL , ASPSR , ASP1 , ASP2 , ASP3 , ASP4 )
C
      IF( ASPN . LT . ASPL ) RETURN
C  YES, REDRAW LINE- THE OLD CONNECTION VIOLATES DIAGONAL DOMINANCE.
C  DRAW LINE DIRECTED FROM I4 TO I3
C  WE HAVE LEFT JE( 3 , IE ) THE SAME SINCE IE IS STILL INTERNAL.
      IDONE = 1
      JE( 1 , IE ) = I4
      JE( 2 , IE ) = I3
      XE( 1 , IE ) = XLL
C
C  ITR IS STILL TO THE RIGHT, ITL TO THE LEFT OF THE NEW LINE IE .
C  FIND THE OTHER DIRECTED LINE SEGMENTS
C
      DO 30 I = 1 , 2
      IM5 = 5 - I
      IF( JE( IM5 , JB ) . NE . ITL ) GO TO 26
      JE( IM5 , JB ) = ITR
 26   CONTINUE
      IF( JE( IM5 , JD ) . NE . ITR ) GO TO 28
      JE( IM5 , JD ) = ITL
 28   CONTINUE
 30   CONTINUE
C
C  RESET JS( 1 - 6 , ITL AND ITR )
C  START BOTH TRIANGLES AT I4 WITH ( AND PUT IN COUNTERCLOCKWISE
C  MANNER)
      JS( 4 , ITR ) = IB
      JS( 5 , ITR ) = IA
      JS( 6 , ITR ) = - IE
      JS( 1 , ITR ) = I4
      JS( 2 , ITR ) = I1
      JS( 3 , ITR ) = I3
      JS( 4 , ITL ) = IE
      JS( 5 , ITL ) = ID
      JS( 6 , ITL ) = IC
```

```
      JS( 1 , ITL ) = I4
      JS( 2 , ITL ) = I3
      JS( 3 , ITL ) = I2
C
      IF( JV( 2 , I1 ) . GT . 0 ) JV( 2 , I1 ) = JA
      IF( JV( 2 , I2 ) . GT . 0 ) JV( 2 , I2 ) = JC
C
      XEL = ( XV( 1 , I3 ) + XV( 1 , I2 ) + XV( 1 , I4 ) ) * THIRD
      YEL = ( XV( 2 , I3 ) + XV( 2 , I2 ) + XV( 2 , I4 ) ) * THIRD
      XER = ( XV( 1 , I3 ) + XV( 1 , I1 ) + XV( 1 , I4 ) ) * THIRD
      YER = ( XV( 2 , I3 ) + XV( 2 , I1 ) + XV( 2 , I4 ) ) * THIRD
C
      DO 92 IR = 1 , MHQ
      HYDV( ITL , IR ) = ( HYDVVV( I3 , IR ) +
     .                     HYDVVV( I2 , IR ) +
     .                     HYDVVV( I4 , IR ) ) * THIRD
C
      HYDV( ITR , IR ) = ( HYDVVV( I3 , IR ) +
     .                     HYDVVV( I1 , IR ) +
     .                     HYDVVV( I4 , IR ) ) * THIRD
 92   CONTINUE
C
      HDUM             = 1. / ( HYDV( ITL , 1 ) + 1.E-12 )
      HYDV( ITL , 2 ) = HYDV( ITL , 2 ) * HDUM
      HYDV( ITL , 3 ) = HYDV( ITL , 3 ) * HDUM
      HYDV( ITL , 4 ) = ( HYDV( ITL , 4 )  -
     .                 .5 * HYDV( ITL , 1 ) *
     .    ( HYDV( ITL , 2 ) * HYDV( ITL , 2 ) +
     .      HYDV( ITL , 3 ) * HYDV( ITL , 3 ) ) ) *
     .                    ( HYDV( ITL , 5 ) - 1. )
C
      HDUM             = 1. / ( HYDV( ITR , 1 ) + 1.E-12 )
      HYDV( ITR , 2 ) = HYDV( ITR , 2 ) * HDUM
      HYDV( ITR , 3 ) = HYDV( ITR , 3 ) * HDUM
      HYDV( ITR , 4 ) = ( HYDV( ITR , 4 )  -
     .                 .5 * HYDV( ITR , 1 ) *
     .    ( HYDV( ITR , 2 ) * HYDV( ITR , 2 ) +
     .      HYDV( ITR , 3 ) * HYDV( ITR , 3 ) ) ) *
     .                    ( HYDV( ITR , 5 ) - 1. )
C
      RGRAD1 = RGRAD( ITL , 1 ) + RGRAD( ITR , 1 )
      RGRAD2 = RGRAD( ITL , 2 ) + RGRAD( ITR , 2 )
      RGRAD( ITL , 1 ) = .5 * RGRAD1
      RGRAD( ITR , 1 ) = .5 * RGRAD1
      RGRAD( ITL , 2 ) = .5 * RGRAD2
      RGRAD( ITR , 2 ) = .5 * RGRAD2
C
      UGRAD1 = UGRAD( ITL , 1 ) + UGRAD( ITR , 1 )
      UGRAD2 = UGRAD( ITL , 2 ) + UGRAD( ITR , 2 )
      UGRAD( ITL , 1 ) = .5 * UGRAD1
      UGRAD( ITR , 1 ) = .5 * UGRAD1
      UGRAD( ITL , 2 ) = .5 * UGRAD2
      UGRAD( ITR , 2 ) = .5 * UGRAD2
C
      VGRAD1 = VGRAD( ITL , 1 ) + VGRAD( ITR , 1 )
      VGRAD2 = VGRAD( ITL , 2 ) + VGRAD( ITR , 2 )
      VGRAD( ITL , 1 ) = .5 * VGRAD1
      VGRAD( ITR , 1 ) = .5 * VGRAD1
      VGRAD( ITL , 2 ) = .5 * VGRAD2
      VGRAD( ITR , 2 ) = .5 * VGRAD2
C
      PGRAD1 = PGRAD( ITL , 1 ) + PGRAD( ITR , 1 )
      PGRAD2 = PGRAD( ITL , 2 ) + PGRAD( ITR , 2 )
      PGRAD( ITL , 1 ) = .5 * PGRAD1
      PGRAD( ITR , 1 ) = .5 * PGRAD1
      PGRAD( ITL , 2 ) = .5 * PGRAD2
      PGRAD( ITR , 2 ) = .5 * PGRAD2
C
      XS( 1 , ITL ) = XEL
      XS( 2 , ITL ) = YEL
      XS( 1 , ITR ) = XER
      XS( 2 , ITR ) = YER
C
      XS( 3 , ITR ) = XSISR
```

```
      XS( 3 , ITL ) = XSISL
C
      SAREA( ITL ) = XSINSL
      SAREA( ITR ) = XSINSR
C
      JEN( 1 ) = JA
      JEN( 2 ) = JB
      JEN( 3 ) = JC
      JEN( 4 ) = JD
      JEN( 5 ) = IE
C
      DO 80 IENN = 1 , 5
      IEN = JEN( IENN )
      JV1 = JE( 1 , IEN )
      JV2 = JE( 2 , IEN )
      AX = XV( 1 , JV2 ) - XV( 1 , JV1 )
      AY = XV( 2 , JV2 ) - XV( 2 , JV1 )
      XEREV = 1. / XE( 1 , IEN )
      XN( IEN ) = AY * XEREV
      YN( IEN ) = - AX * XEREV
      ISSR = JE( 4 , IEN )
      ISSL = JE( 3 , IEN )
      IJE5 = JE( 5 , IEN )
      IF( IJE5 . NE . 0 ) THEN
C
      AA = XV( 1 , JV2 ) - XV( 1 , JV1 )
      BB = XV( 2 , JV2 ) - XV( 2 , JV1 )
      XEL = XS( 1 , ISSL )
      YEL = XS( 2 , ISSL )
      CC = XEL - XV( 1 , JV1 )
      DD = YEL - XV( 2 , JV1 )
      EE = ( AA * CC + BB * DD ) * XEREV * XEREV
      XER = XV( 1 , JV1 ) + AA * EE
      YER = XV( 2 , JV1 ) + BB * EE
      AX = XER - XEL
      AY = YER - YEL
      XE( 2 , IEN ) = SQRT( AX * AX + AY * AY )
      XEREV = 1. / XE( 2 , IEN )
      XXN( IEN ) = AX * XEREV
      YYN( IEN ) = AY * XEREV
      XE( 2 , IEN ) = 2. * XE( 2 , IEN )
      XYMIDL( IEN ) = .5
      XMIDL( IEN ) = XER
      YMIDL( IEN ) = YER
C
      ELSE
C
      XER = XS( 1 , ISSR )
      YER = XS( 2 , ISSR )
      XEL = XS( 1 , ISSL )
      YEL = XS( 2 , ISSL )
C
      AA = XV( 1 , JV2 ) - XV( 1 , JV1 )
      BB = XV( 2 , JV2 ) - XV( 2 , JV1 )
      CC = XEL - XER
      DD = YEL - YER
      ACA = XER - XV( 1 , JV1 )
      DBD = YER - XV( 2 , JV1 )
      EE = ( ACA * DD - DBD * CC ) / ( AA * DD - BB * CC )
      XMIDL( IEN ) = XV( 1 , JV1 ) + AA * EE
      YMIDL( IEN ) = XV( 2 , JV1 ) + BB * EE
C
      XEMID = XMIDL( IEN ) - XEL
      YEMID = YMIDL( IEN ) - YEL
C
      AX = XER - XEL
      AY = YER - YEL
      XE( 2 , IEN ) = SQRT( AX * AX + AY * AY )
      XEREV = 1. / XE( 2 , IEN )
      XXN( IEN ) = AX * XEREV
      YYN( IEN ) = AY * XEREV
C
      XYMIDL( IEN ) = SQRT( XEMID * XEMID + YEMID * YEMID ) * XEREV
C
```

```
         END IF
C
   80    CONTINUE
C
           RETURN
C
  999      WRITE (6,1000) IE
C
C --- EXIT POINT FROM SUBROUTINE ------------------------------------
C
C        ------
         RETURN
C        ------
C
C --- FORMATS -------------------------------------------------------
C
 1000 FORMAT('0ITS ABOUT TO BOMB--RECNC ON EDGE ',I5)
C
C        ---
         END
```



```
      SUBROUTINE EOS (RRR,EEE,N,GAMMA)
C
C-------------------------------------------------------------I
C                                                             I
C     AIR IS ASSUMED TO BE CALORICALLY IMPERFECT, THERMALLY   |
C     PERFECT. THEREFORE, INCLUDE IMPERFECTIONS VIA A VARIABLE |
C     GAMMA DEPENDENTON DENSITY AND INTERNAL ENERGY.          |
C     THIS ROUTINE PERFORMS A TABLE LOOK UP FOR GAMMA.        |
C                                                             I
C-------------------------------------------------------------I
C
C     INPUT VARIBLE DEFINITIONS.
C     RRR = MASS DENSITY
C     EEE = INTERNAL ENERGY PER UNIT VOLUME
C           (CONVERTED FOR INTERNAL *CALL TO ENERGY PER UNIT MASS)
C     N   = NUMBER OF ENTRIES IN ARRAYS RRR & EEE
C
      PARAMETER (M = 64 )
C
      DIMENSION RRR(N), EEE(N), GAMMA(N)
      DIMENSION T11(M), T12(M), T21(M), T22(M), RHO(M), E(M)
      DIMENSION OMP(M), Q(M), I(M), J(M)
      DIMENSION G1(168),G2(112),G3(112),G4(112),G5(112),
     !          G6(112),G7(112),GF(840)
C
C     NOTE:  THE TABLE LOOK UP TREATS ARRAY GF AS THOUGH IT
C     WERE DIMENSIONED (8,105).
C
      EQUIVALENCE (G1(1),GF(  1)), (G2(1),GF(169)), (G3(1),GF(281)),
     !            (G4(1),GF(393)), (G5(1),GF(505)), (G6(1),GF(617)),
     !            (G7(1),GF(729))
C
      DATA XL16E  /2.772588722239774483568908181041479110717773437S/
C-----------------------------------------------------
C     G = GAMMA - 1.0 IS STORED FOR 32 BIT WORD MACHINES IN POWERS OF
C     16 ACROSS FOR MASS DENSITY VARIATION AND INTERMEDIATE VALUES
C     1 - 16 FOR POWERS OF 16 VERTICALLY WHICH REPRESENT THE INTERNAL
C     ENERGY VARIATION.
C
C     16**(2)  .GE. RHO .GE. 16**(-6)
C     16**(15) .GE.  E  .GE. 16**(8)
C-----------------------------------------------------
      DATA G1 /8*.4222,8*.4152,8*.4110,8*.4081,8*.4058,8*.4040,
     !         8*.4024,8*.4011,8*.3998,8*.3988,8*.3978,8*.3969,
     !         8*.3961,8*.3953,8*.3935,8*.3918,
     !        .3723,.3715,.3707,.3699,.3690,.3680,.3663,.3637,
     !        .3555,.3538,.3522,.3502,.3476,.3430,.3344,.3238,
     !        .3370,.3370,.3370,.3364,.3347,.3277,.3099,.2885,
     !        .3257,.3227,.3201,.3134,.3062,.3014,.2884,.2591,
     !        .3166,.3110,.3063,.2946,.2831,.2783,.2677,.2358/
      DATA G2/.3111,.3006,.2940,.2787,.2635,.2588,.2502,.2236,
```

```
     !           .3075,.2906,.2810,.2665,.2466,.2418,.2350,.2131,
     !           .3043,.2819,.2695,.2554,.2317,.2269,.2216,.2038,
     !           .2929,.2740,.2593,.2455,.2206,.2136,.2097,.1955,
     !           .2840,.2672,.2500,.2366,.2166,.2015,.1988,.1879,
     !           .2764,.2611,.2429,.2285,.2125,.1890,.1890,.1811,
     !           .2714,.2555,.2384,.2210,.2079,.1818,.1799,.1747,
     !           .2669,.2504,.2343,.2141,.2037,.1822,.1709,.1689,
     !           .2624,.2473,.2304,.2096,.1998,.1828,.1684,.1639,
     !           .2599,.2446,.2268,.2087,.1961,.1834,.1673,.1601,
     !           .2401,.2191,.1972,.1775,.1592,.1444,.1358,.1203,
     !           .2002,.1960,.1749,.1536,.1376,.1252,.1107,.1044,
     !           .1911,.1829,.1633,.1420,.1266,.1101,.1012,.0933,
     !           .1950,.1781,.1566,.1415,.1241,.1118,.1009,.0948/
      DATA G3/.2001,.1789,.1594,.1443,.1306,.1189,.1095,.1013,
     !           .2040,.1826,.1657,.1494,.1338,.1177,.1081,.0980,
     !           .2034,.1854,.1683,.1497,.1322,.1169,.1051,.0946,
     !           .1969,.1855,.1685,.1487,.1304,.1149,.1024,.0916,
     !           .1899,.1837,.1677,.1475,.1287,.1126,.1002,.0900,
     !           .1841,.1817,.1667,.1464,.1272,.1109,.0983,.0888,
     !           .1800,.1800,.1659,.1455,.1262,.1097,.0965,.0878,
     !           .1779,.1787,.1657,.1450,.1254,.1087,.0949,.0868,
     !           .1773,.1778,.1656,.1447,.1250,.1080,.0939,.0859,
     !           .1783,.1778,.1658,.1448,.1248,.1076,.0933,.0851,
     !           .1808,.1781,.1667,.1451,.1248,.1074,.0930,.0843,
     !           .2134,.2040,.1978,.1782,.1565,.1368,.1206,.1074,
     !           .2210,.2072,.1957,.1739,.1516,.1312,.1137,.1000,
     !           .2245,.2109,.1989,.1772,.1563,.1390,.1247,.1133/
      DATA G4/.2299,.2132,.2017,.1795,.1579,.1384,.1221,.1090,
     !           .2350,.2157,.2023,.1798,.1575,.1370,.1197,.1057,
     !           .2397,.2194,.2034,.1796,.1572,.1372,.1205,.1070,
     !           .2452,.2227,.2050,.1805,.1576,.1379,.1236,.1118,
     !           .2510,.2256,.2069,.1814,.1581,.1383,.1231,.1103,
     !           .2560,.2282,.2091,.1822,.1585,.1385,.1226,.1083,
     !           .2605,.2312,.2111,.1829,.1588,.1386,.1222,.1070,
     !           .2677,.2358,.2129,.1836,.1592,.1386,.1218,.1071,
     !           .2759,.2403,.2145,.1857,.1598,.1389,.1219,.1078,
     !           .2834,.2445,.2160,.1878,.1603,.1394,.1223,.1084,
     !           .2905,.2484,.2175,.1898,.1613,.1399,.1226,.1090,
     !           .2963,.2531,.2199,.1918,.1625,.1407,.1230,.1096,
     !           .4323,.3582,.3109,.2889,.2803,.2706,.2410,.2224,
     !           .4610,.4026,.3624,.3212,.2926,.2551,.2375,.2015/
      DATA G5/.4199,.3837,.3401,.2979,.2623,.2318,.2108,.1854,
     !           .3924,.3642,.3194,.2760,.2427,.2157,.1902,.1721,
     !           .3794,.3479,.3025,.2673,.2311,.2019,.1842,.1613,
     !           .3674,.3448,.2961,.2593,.2255,.1994,.1785,.1594,
     !           .3573,.3443,.2910,.2517,.2293,.2006,.1843,.1679,
     !           .3661,.3438,.2935,.2597,.2336,.2225,.2143,.2116,
     !           .3674,.3435,.3080,.2728,.2606,.2577,.2573,.2573,
     !           .3685,.3453,.3210,.3014,.2942,.2933,.2932,.2932,
     !           .3814,.3612,.3341,.3276,.3257,.3253,.3252,.3252,
     !           .3903,.3752,.3570,.3522,.3513,.3510,.3506,.3496,
     !           .4012,.3899,.3782,.3751,.3743,.3741,.3734,.3713,
     !           .4155,.4057,.3956,.3930,.3920,.3913,.3907,.3890,
     !           .4290,.4205,.4118,.4092,.4077,.4065,.4059,.4047,
     !           .5411,.5385,.5359,.5353,.5351,.5350,.5350,.5350/
      DATA G6/.5823,.5812,.5801,.5797,.5796,.5797,.5797,.5797,
     !           .6096,.6090,.6085,.6082,.6082,.6083,.6083,.6083,
     !           .6308,.6306,.6305,.6303,.6303,.6305,.6305,.6305,
     !           .6481,.6483,.6485,.6483,.6484,.6486,.6487,.6487,
     !           .6627,.6632,.6637,.6636,.6637,.6640,.6640,.6640,
     !           .6754,.6761,.6769,.6768,.6770,.6773,.6773,.6773,
     !           .6866,.6875,.6885,.6884,.6886,.6890,.6890,.6890,
     !           .6966,.6977,.6989,.6989,.6991,.6995,.6995,.6995,
     !           .7056,.7070,.7083,.7083,.7085,.7090,.7090,.7090,
     !           .7139,.7154,.7169,.7169,.7172,.7176,.7177,.7177,
     !           .7214,.7231,.7248,.7248,.7251,.7256,.7256,.7256,
     !           .7285,.7303,.7321,.7321,.7325,.7330,.7330,.7330,
     !           .7350,.7370,.7390,.7390,.7393,.7398,.7399,.7399,
     !           .7411,.7432,.7453,.7454,.7457,.7463,.7463,.7463/
      DATA G7/.8069,.8103,.8138,.8139,.8145,.8152,.8153,.8153,
     !           .8454,.8496,.8538,.8540,.8547,.8556,.8557,.8557,
     !           .8727,.8774,.8822,.8825,.8832,.8842,.8843,.8843,
     !           .8938,.8990,.9042,.9046,.9054,.9064,.9065,.9065,
     !           .9111,.9166,.9222,.9226,.9235,.9246,.9247,.9247,
```

```
     !        .9258,.9316,.9374,.9379,.9387,.9399,.9400,.9400,
     !        .9384,.9445,.9506,.9511,.9520,.9532,.9533,.9533,
     !        .9496,.9559,.9622,.9627,.9637,.9649,.9650,.9650,
     !        .9596,.9661,.9727,.9731,.9741,.9754,.9755,.9755,
     !        .9686,.9753,.9821,.9826,.9836,.9849,.9850,.9850,
     !        .9769,.9837,.9906,.9912,.9922,.9936,.9937,.9937,
     !        .9845,.9915,.9986,.9991,.9999,.9999,.9999,.9999,
     !        .9915,.9987,.9999,.9999,.9999,.9999,.9999,.9999,
     !        .9981,.9999,.9999,.9999,.9999,.9999,.9999,.9999/
C-----------------------------------------------------------------
C     REAL AIR EOS, TABLE LOOKUP ON GILMORE DATA.  (NO TEMP. MODEL)
C     TO AVOID COSTLY LOGARITHMIC FUNCTIONS THE TABLE "G" IS STORED IN A
C     FORM SO THAT THE HEXADECIMAL WORD STRUCTURE OF A 32 BIT MACHINE
C     MAY BE EXPLOITED.
C     THIS LOGIC MAY BE TRANSFERED TO OTHER MACHINES BY RECALCULATING
C     THE TABLE "G" APPROPRIATE TO THE WORD ARCITECTURE OF THAT MACHINE.
C     MACHINE DEPENDENT FUNCTIONS AND KEY NUMBERS MUST ALSO BE CHANGED.
C-----------------------------------------------------------------
      RL16E = 1./XL16E
      IST = 0
      NR  = N
C
   10 CONTINUE
      NST = MIN0(NR,M)
C
      DO 20 IRE=1,NST
      RHO(IRE) = .774413*RRR(IST+IRE)
      E(IRE) = AMAX1(3.e8,10000.*EEE(IST+IRE)/RRR(IST+IRE))
C
C     CALCULATE MASS DENSITY VARIATION INDEX "I".
C
      TEM = ALOG(RHO(IRE))*RL16E + 500.0
      I(IRE)   = AINT(TEM)
      OMP(IRE) = TEM - FLOAT(I(IRE))
      I(IRE)   = 502 - I(IRE)
      I(IRE)   = MAX0(I(IRE),1)
C
C     CALCULATE INTERNAL ENERGY VARIATION INDEX "J".
C
      TEM = ALOG(E(IRE))*RL16E
      JCY = AINT(TEM)
      TEM = TEM - FLOAT(JCY)
      TEM = EXP(XL16E*TEM)
      JCY = JCY - 7
      JS  = AINT(TEM)
      Q(IRE)   = TEM - FLOAT(JS)
      J(IRE)   = JS + 15*JCY
      J(IRE)   = MIN0(J(IRE),104)
      J(IRE)   = I(IRE) + 8*J(IRE)
      I(IRE)   = J(IRE) - 8
   20 CONTINUE
C
      DO 30 IRE=1,NST
      T11(IRE) = GF(I(IRE))
      T21(IRE) = GF(I(IRE)+1)
      T12(IRE) = GF(J(IRE))
      T22(IRE) = GF(J(IRE)+1)
   30 CONTINUE
C
C     CALCULATE GAMMA BY LINEAR INTERPOLATION.
C
      DO 40 IRE=1,NST
      T12(IRE) = T12(IRE) - T11(IRE)
      T22(IRE) = T22(IRE) - T21(IRE)
      GAMMA(IST+IRE) =       OMP(IRE) *(T11(IRE) + Q(IRE)*T12(IRE))
     !              + (1. - OMP(IRE))*(T21(IRE) + Q(IRE)*T22(IRE))
     !              +  1.
   40 CONTINUE
C
      NR  = NR - NST
      IST = IST + NST
      IF(NR.GT.0) GO TO 10
C
C --- EXIT POINT FROM SUBROUTINE ---------------------------------------
```

```
C
C     ------
      RETURN
C     ------
C
C     ---
      END
```

Thu Jul 1 14:16:08 1993 adaphd.f SUBROUTINE LIFTDR

```
      SUBROUTINE LIFTDR
C
      include     'cmsh00.h'
      include     'chyd00.h'
      include     'cint00.h'
      include     'cphs10.h'
      include     'cphs20.h'
      REAL PRESS(1000),DYNPRS(1000),XLOCAT(1000),YLOCAT(1000)
C
        XLIFT = 0.
        XDRAG = 0.
        XMOMN = 0.
        UINVR = 2. / UVIN / UVIN / RIN
        XYU = COS( ALPHA )
        XYV = SIN( ALPHA )
        NBB = 0
           DO 210 IE = 1 , NE
       IJE5 = JE( 5 , IE )
       IF( IJE5 . EQ . 5 ) THEN
           NBB = NBB + 1
           IV1 = JE( 1 , IE )
           IV2 = JE( 2 , IE )
           ISL = JE( 3 , IE )
           PRES = HYDV( ISL , 4 ) - PINL
           PRESS( NBB ) = PRES
           XLIFT = XLIFT + PRES * XE( 1 , IE ) *
     .                  ( - XN( IE ) * XYV + YN( IE ) * XYU )
           XDRAG = XDRAG + PRES * XE( 1 , IE ) *
     .                  ( XN( IE ) * XYU + YN( IE ) * XYV )
           XLOCAT( NBB ) = .5 * ( XV( 1 , IV1 ) + XV( 1 , IV2 ) )
           XXV = XLOCAT( NBB )
           YLOCAT( NBB ) = .5 * ( XV( 2 , IV1 ) + XV( 2 , IV2 ) )
           YYV = YLOCAT( NBB )
           XMOMN = XMOMN + PRES * XE( 1 , IE ) *
     .                  ( XN( IE ) * XXV - YN( IE ) * YYV )
C
           END IF
C
  210      CONTINUE
C
        XLIFT = XLIFT * UINVR
        XDRAG = XDRAG * UINVR
        XMOMN = XMOMN * UINVR
        WRITE (4) NBB,(XLOCAT(KK),YLOCAT(KK),PRESS(KK),KK=1,NBB)
        WRITE (9) XLIFT,XDRAG,XMOMN,XMCHIN,ALFA
        PRINT *, XLIFT,XDRAG,XMOMN,XMCHIN,ALFA
C
C --- EXIT POINT FROM SUBROUTINE -------------------------------------
C
C     ------
      RETURN
C     ------
C
C     ---
      END
```

```
      SUBROUTINE VERDELT( KSD , INDCTR , NIDUMP , JJTRIG , IITRIG )
C
C-------------------------------------------------------------I
C                                                             I
C     VERDEL FORCE DELETION OF CELL NUMBER KSD                I
C                                                             I
C-------------------------------------------------------------I
C
      IMPLICIT REAL (A-H,O-Z)
C
      include     'cmsh00.h'
      include     'chyd00.h'
      include     'cint00.h'
      include     'cphs10.h'
      include     'cphs20.h'
C
        INTEGER JUV(MEM),JUE(MEM),JUS(MEM)
        INTEGER IUV(MEM),IUE(MEM),IUS(MEM)
        INTEGER IITRIG(200)
C
      EQUIVALENCE (PR,JUV)
      EQUIVALENCE (UR,JUE)
      EQUIVALENCE (VR,JUS)
      EQUIVALENCE (PL,IUV)
      EQUIVALENCE (UL,IUE)
      EQUIVALENCE (VL,IUS)
C
C   JUV( IVV ) >> NV
C   IUV( NV ) >> IVV
C
C          DO *** KI = 1 , JVDELT
C          IVM = NVDELT( KI )
C          JVM = IVDELT( KI )
C          JUV( IVM ) = JVM
C          IUV( JVM ) = IVM
C
C          DO *** KI= 1 , IETRIG
C          IEM = NECRSS( KI )
C          JEM = IECRSS( KI )
C          JUE( IEM ) = JEM
C          IUE( JEM ) = IEM
C
C          DO *** KI = 1 , ITRIG
C          ISM = NSCRSS( KI )
C          JSM = ISCRSS( KI )
C          JUS( ISM ) = JSM
C          IUS( JSM ) = ISM
C
         FLUXPP = .00001 * HYDMOM( 4 )
         FLUXUU = .00001 * HYDMOM( 2 )
         FLUXRR = .00001 * HYDMOM( 1 )
         AREVGG = AREDEL * SAREVG
         XYLONG = 0.
         XYSHRT = 10000000.
         XYLNGT = 0.
C
         KV1 = JS( 1 , KSD )
         KV2 = JS( 2 , KSD )
         KV3 = JS( 3 , KSD )
         JKV1 = JV( 2 , KV1 )
         JKV2 = JV( 2 , KV2 )
         JKV3 = JV( 2 , KV3 )
         KE1 = JS( 4 , KSD )
         KE2 = JS( 5 , KSD )
         KE3 = JS( 6 , KSD )
         IKE1 = IABS( KE1 )
         IKE2 = IABS( KE2 )
         IKE3 = IABS( KE3 )
         IJE51 = JE( 5 , IKE1 )
         IJE52 = JE( 5 , IKE2 )
         IJE53 = JE( 5 , IKE3 )
         IKKE = 0
         IF( IJE53 . NE . 0 . AND . JKV2 . LT . 0 ) THEN
```

```
            IKKE = 4
            IEIN1 = - JKV1
            IEIN2 = - JKV2
            IKKE1 = IKE3
            KKE1 = KE3
            KKV1 = KV3
            IKKE2 = IKE1
            KKE2 = KE1
            KKV2 = KV1
            IKKE3 = IKE2
            KKE3 = KE2
            KKV3 = KV2
C
            ELSE IF( IJE52 . NE . 0 . AND . JKV1 . LT . 0 ) THEN
            IKKE = 4
            IEIN1 = - JKV3
            IEIN2 = - JKV1
            IKKE1 = IKE2
            KKE1 = KE2
            KKV1 = KV2
            IKKE2 = IKE3
            KKE2 = KE3
            KKV2 = KV3
            IKKE3 = IKE1
            KKE3 = KE1
            KKV3 = KV1
C
            ELSE IF( IJE51 . NE . 0 . AND . JKV3 . LT . 0 ) THEN
            IKKE = 4
            IEIN1 = - JKV2
            IEIN2 = - JKV3
            IKKE1 = IKE1
            KKE1 = KE1
            KKV1 = KV1
            IKKE2 = IKE2
            KKE2 = KE2
            KKV2 = KV2
            IKKE3 = IKE3
            KKE3 = KE3
            KKV3 = KV3
C
            ELSE IF( IJE53 . EQ . 0 . AND . JKV3 . LT . 0 . AND .
     .                                JKV1 . LT . 0 ) THEN
            IKKE = 3
            IEIN1 = - JKV3
            IEIN2 = - JKV1
            IKKE1 = IKE3
            KKE1 = KE3
            KKV1 = KV3
            IKKE2 = IKE1
            KKE2 = KE1
            KKV2 = KV1
            IKKE3 = IKE2
            KKE3 = KE2
            KKV3 = KV2
C
            ELSE IF( IJE52 . EQ . 0 . AND . JKV2 . LT . 0 . AND .
     .                                JKV3 . LT . 0 ) THEN
            IKKE = 3
            IEIN1 = - JKV2
            IEIN2 = - JKV3
            IKKE1 = IKE2
            KKE1 = KE2
            KKV1 = KV2
            IKKE2 = IKE3
            KKE2 = KE3
            KKV2 = KV3
            IKKE3 = IKE1
            KKE3 = KE1
            KKV3 = KV1
C
            ELSE IF( IJE51 . EQ . 0 . AND . JKV1 . LT . 0 . AND .
     .                                JKV2 . LT . 0 ) THEN
            IKKE = 3
```

```
            IEIN1 = - JKV1
            IEIN2 = - JKV2
            IKKE1 = IKE1
            KKE1 = KE1
            KKV1 = KV1
            IKKE2 = IKE2
            KKE2 = KE2
            KKV2 = KV2
            IKKE3 = IKE3
            KKE3 = KE3
            KKV3 = KV3
C
            ELSE IF( IJE53 . NE . 0 ) THEN
            IKKE = 1
            IEIN = - JKV1
            IKKE1 = IKE3
            KKE1 = KE3
            KKV1 = KV3
            IKKE2 = IKE1
            KKE2 = KE1
            KKV2 = KV1
            KKE3 = KE2
            IKKE3 = IKE2
            KKV3 = KV2
C
            ELSE IF( IJE52 . NE . 0 ) THEN
            IKKE = 1
            IEIN = - JKV3
            IKKE1 = IKE2
            KKE1 = KE2
            KKV1 = KV2
            IKKE2 = IKE3
            KKE2 = KE3
            KKV2 = KV3
            IKKE3 = IKE1
            KKE3 = KE1
            KKV3 = KV1
C
            ELSE IF( IJE51 . NE . 0 ) THEN
            IKKE = 1
            IEIN = - JKV2
            IKKE1 = IKE1
            KKE1 = KE1
            KKV1 = KV1
            IKKE2 = IKE2
            KKE2 = KE2
            KKV2 = KV2
            IKKE3 = IKE3
            KKE3 = KE3
            KKV3 = KV3
C
            ELSE IF( JKV3 . LT . 0 ) THEN
            IKKE = 2
            IEIN = - JKV3
            IKKE1 = IKE3
            KKE1 = KE3
            KKV1 = KV3
            IKKE2 = IKE1
            KKE2 = KE1
            KKV2 = KV1
            IKKE3 = IKE2
            KKE3 = KE2
            KKV3 = KV2
C
            ELSE IF( JKV2 . LT . 0 ) THEN
            IKKE = 2
            IEIN = - JKV2
            IKKE1 = IKE2
            KKE1 = KE2
            KKV1 = KV2
            IKKE2 = IKE3
            KKE2 = KE3
            KKV2 = KV3
            IKKE3 = IKE1
```

```
            KKE3 = KE1
            KKV3 = KV1
C
            ELSE IF( JKV1 . LT . 0 ) THEN
            IKKE = 2
            IEIN = - JKV1
            IKKE1 = IKE1
            KKE1 = KE1
            KKV1 = KV1
            IKKE2 = IKE2
            KKE2 = KE2
            KKV2 = KV2
            IKKE3 = IKE3
            KKE3 = KE3
            KKV3 = KV3
            END IF
C
            IF( IKKE . EQ . 4 ) THEN
            JV1 = JE( 1 , IEIN2 )
            JV2 = JE( 2 , IEIN2 )
            JJV3 = JE( 1 , IKKE3 )
            JJV4 = JE( 2 , IKKE3 )
            IF( JJV3 . EQ . JV1 ) THEN
            JV3 = JJV3
            JV4 = JJV4
            ELSE
            JV3 = JJV4
            JV4 = JJV3
            END IF
            XA = XV( 1 , JV2 ) - XV( 1 , JV1 )
            YA = XV( 2 , JV2 ) - XV( 2 , JV1 )
            XB = XV( 1 , JV4 ) - XV( 1 , JV3 )
            YB = XV( 2 , JV4 ) - XV( 2 , JV3 )
            AB = XA * XB + YA * YB
            IF( AB . GT . 0. ) IKKE = 5
            END IF
C
C    IJTRIG NUMBER OF CIRCUMFERENCE EDGES AROUND VOID
C    ITRIG NUMBER OF TRIANGLES TO BE DELETED
C    IETRIG NUMBER OF EDGES TO BE DELETED
C    JVDELT NUMBER OF VERTICES TO BE DELETED
C
C    IVDELT(*) SEQUENCE OF VERTICES TO BE DELETED
C    ISCRSS(*) SEQUENCE OF TRIANGLES TO BE DELETED
C    IECRSS(*) SEQUENCE OF EDGES TO BE DELETED
C
            IF( JV( 1 , KV1 ) . EQ . 3 ) RETURN
            IF( JV( 1 , KV2 ) . EQ . 3 ) RETURN
            IF( JV( 1 , KV3 ) . EQ . 3 ) RETURN
            IJTRIG = 0
            ITRIG = 0
            IETRIG = 0
            JVDELT = 0
            JLOOP = 0
C
            IF( IKKE . EQ . 0 ) THEN
C
C     THE TRIANGLE TO BE DELETED IS INTIRELY IN THE DOMAIN OF COMPUTATION .
C     THE FIRST LOOP IS AROUND VERTEX KV1 .
C
            IVV = KV1
            IE = IKE3
            IV1 = JE( 1 , IE )
            IF( IV1 . EQ . IVV ) THEN
            ISI = JE( 3 , IE )
            ELSE
            ISI = JE( 4 , IE )
            END IF
            IS = ISI
C
  110       CONTINUE
C
            ITRIG = ITRIG + 1
            ISCRSS( ITRIG ) = IS
```

```
C
            IF( ITRIG . EQ . 2 ) THEN
            IJTRIG = 0
            IETRIG = IETRIG + 1
            IECRSS( IETRIG ) = IEIB
            END IF
C
            IETRIG = IETRIG + 1
            IECRSS( IETRIG ) = IE
C
            IF(
     .         HYDFLX( IS , 4 ) . GT . FLUXPP . OR .
     .         HYDFLX( IS , 2 ) . GT . FLUXUU . OR .
     .         HYDFLX( IS , 1 ) . GT . FLUXRR . OR .
     .         KSDELT( IS ) . GT . NIDUMP . OR .
     .         XS( 3 , IS ) . GT . AREVGG ) THEN
            INDCTR = 3
            RETURN
            END IF
C
            DO 120 IR = 1 , 3
            JR = MOD( IR , 3 ) + 1
            IEA = IABS( JS( JR + 3 , IS ) )
            IF( IEA . EQ . IE ) THEN
            IIR = MOD( JR , 3 ) + 4
            IEI = JS( IIR , IS )
            IEIB = IABS( IEI )
            XEIEB = XE( 1 , IEIB )
            XYLNGT = XYLNGT + XEIEB
            IF( XYLONG . LT . XEIEB ) XYLONG = XEIEB
            IF( XYSHRT . GT . XEIEB ) XYSHRT = XEIEB
            IJTRIG = IJTRIG + 1
            IICOLR( IJTRIG ) = IEI
            JJR = MOD( JR + 1 , 3 ) + 4
            IER = IABS( JS( JJR , IS ) )
C
            IV1 = JE( 1 , IER )
            IF( IV1 . EQ . IVV ) THEN
            ISR = JE( 3 , IER )
            ELSE
            ISR = JE( 4 , IER )
            END IF
            END IF
C
  120       CONTINUE
C
            IF( ISR . NE . ISI ) THEN
            IS = ISR
            IE = IER
            GO TO 110
            END IF
C
            IETRIG = IETRIG + 1
            IECRSS( IETRIG ) = IKE2
            IJTRIG = IJTRIG - 2
C
C     FIRST LOOP SUROUNDING KV1 IS DONE, SECOND LOOP OVER KV2 START .
C
            IVV = KV2
            IE = IABS( IICOLR( IJTRIG + 1 ) )
            IV1 = JE( 1 , IE )
            IF( IV1 . EQ . IVV ) THEN
            ISI = JE( 3 , IE )
            ELSE
            ISI = JE( 4 , IE )
            END IF
            IS = ISI
C
            ILOOP = 0
  130       CONTINUE
            JDOUBL = IABS( IICOLR( IJTRIG ) )
C
            ITRIG = ITRIG + 1
            ISCRSS( ITRIG ) = IS
```

```
      IETRIG = IETRIG + 1
      IECRSS( IETRIG ) = IE
C
      IF(
     .    HYDFLX( IS , 4 ) . GT . FLUXPP . OR .
     .    HYDFLX( IS , 2 ) . GT . FLUXUU . OR .
     .    HYDFLX( IS , 1 ) . GT . FLUXRR . OR .
     .    KSDELT( IS ) . GT . NIDUMP . OR .
     .    XS( 3 , IS ) . GT . AREVGG ) THEN
      INDCTR = 3
      RETURN
      END IF
C
      DO 140 IR = 1 , 3
      JR = MOD( IR , 3 ) + 1
      IEA = IABS( JS( JR + 3 , IS ) )
      IF( IEA . EQ . IE ) THEN
      IIR = MOD( JR , 3 ) + 4
      IEI = JS( IIR , IS )
      IEIB = IABS( IEI )
      XEIEB = XE( 1 , IEIB )
      XYLNGT = XYLNGT + XEIEB
      IF( XYLONG . LT . XEIEB ) XYLONG = XEIFB
      IF( XYSHRT . GT . XEIEB ) XYSHRT = XEIEB
      ILOOP = ILOOP + 1
      IF( ILOOP . EQ . 1 . AND . JDOUBL . EQ . IEIB ) THEN
      JLOOP = 1
      IETRIG = IETRIG + 1
      IECRSS( IETRIG ) = JDOUBL
      IJTRIG = IJTRIG - 1
      IF( IEI . GT . 0 ) THEN
      JKVV = JE( 1 , IEIB )
      ELSE
      JKVV = JE( 2 , IEIB )
      END IF
      JVDELT = JVDELT + 1
      IVDELT( JVDELT ) = JKVV
      ILOOP = 0
      ELSE
      IJTRIG = IJTRIG + 1
      IICOLR( IJTRIG ) = IEI
      END IF
      JJR = MOD( JR + 1 , 3 ) + 4
      IER = IABS( JS( JJR , IS ) )
C
      IV1 = JE( 1 , IER )
      IF( IV1 . EQ . IVV ) THEN
      ISR = JE( 3 , IER )
      ELSE
      ISR = JE( 4 , IER )
      END IF
      END IF
C
140   CONTINUE
C
      IF( IER . NE . IKE2 ) THEN
      IS = ISR
      IE = IER
      GO TO 130
      END IF
      IJTRIG = IJTRIG - 1
C
C   SECOND LOOP SUROUNDING KV2 IS DONE, THIRD LOOP OVER KV3 START .
C
      KET = IECRSS( 2 )
      IVV = KV3
      IE = IABS( IICOLR( IJTRIG + 1 ) )
      IF( IE . EQ . KET ) THEN
      JLOOP = 2
C
150   CONTINUE
      IKET = IICOLR( 1 )
      KKET = IABS( IKET )
      JKET = IABS( IICOLR( IJTRIG ) )
```

```
                  IF( JKET . EQ . KKET ) THEN
                  JLOOP = 3
                  IF( IKET . GT . 0 ) THEN
                  JKVV = JE( 1 , KKET )
                  ELSE
                  JKVV = JE( 2 , KKET )
                  END IF
                  JVDELT = JVDELT + 1
                  IVDELT( JVDELT ) = JKVV
                  DO 160 KK = 2 , IJTRIG
                  IICOLR( KK - 1 ) = IICOLR( KK )
      160         CONTINUE
                  IJTRIG = IJTRIG - 2
                  IETRIG = IETRIG + 1
                  IECRSS( IETRIG ) = KKET
                  GO TO 150
                  END IF
                  GO TO 170
                  END IF
                  IV1 = JE( 1 , IE )
                  IF( IV1 . EQ . IVV ) THEN
                  ISI = JE( 3 , IE )
                  ELSE
                  ISI = JE( 4 , IE )
                  END IF
                  IS = ISI
C
                  ILOOP = 0
      180         CONTINUE
                  KDOUBL = IABS( IICOLR( IJTRIG ) )
C
                  ITRIG = ITRIG + 1
                  ISCRSS( ITRIG ) = IS
                  IETRIG = IETRIG + 1
                  IECRSS( IETRIG ) = IE
C
                  IF(
     .               HYDFLX( IS , 4 ) . GT . FLUXPP . OR .
     .               HYDFLX( IS , 2 ) . GT . FLUXUU . OR .
     .               HYDFLX( IS , 1 ) . GT . FLUXRR . OR .
     .               KSDELT( IS ) . GT . NIDUMP . OR .
     .               XS( 3 , IS ) . GT . AREVGG ) THEN
                  INDCTR = 3
                  RETURN
                  END IF
C
                  DO 190 IR = 1 , 3
                  JR = MOD( IR , 3 ) + 1
                  IEA = IABS( JS( JR + 3 , IS ) )
                  IF( IEA . EQ . IE ) THEN
                  IIR = MOD( JR , 3 ) + 4
                  IEI = JS( IIR , IS )
                  IEIB = IABS( IEI )
                  XEIEB = XE( 1 , IEIB )
                  XYLNGT = XYLNGT + XEIEB
                  IF( XYLONG . LT . XEIEB ) XYLONG = XEIEB
                  IF( XYSHRT . GT . XEIEB ) XYSHRT = XEIEB
                  ILOOP = ILOOP + 1
                  IF( ILOOP . EQ . 1 . AND . KDOUBL . EQ . IEIB ) THEN
                  JLOOP = 4
                  IETRIG = IETRIG + 1
                  IECRSS( IETRIG ) = KDOUBL
                  IJTRIG = IJTRIG - 1
                  IF( IEI . GT . 0 ) THEN
                  JKVV = JE( 1 , IEIB )
                  ELSE
                  JKVV = JE( 2 , IEIB )
                  END IF
                  JVDELT = JVDELT + 1
                  IVDELT( JVDELT ) = JKVV
                  ILOOP = 0
                  ELSE
                  IJTRIG = IJTRIG + 1
                  IICOLR( IJTRIG ) = IEI
```

```
                 END IF
                 JJR = MOD( JR + 1 , 3 ) + 4
                 IER = IABS( JS( JJR , IS ) )
C
                 IV1 = JE( 1 , IER )
                 IF( IV1 . EQ . IVV ) THEN
                 ISR = JE( 3 , IER ) .
                 ELSE
                 ISR = JE( 4 , IER )
                 END IF
                 END IF
C
190              CONTINUE
C
                 IF( IER . NE . KET ) THEN
                 IS = ISR
                 IE = IER
                 GO TO 180
                 END IF
C
200              CONTINUE
                 IKET = IICOLR( 1 )
                 KKET = IABS( IKET )
                 JKET = IABS( IICOLR( IJTRIG ) )
                 IF( JKET . EQ . KKET ) THEN
                 JLOOP = 5
                 IF( IKET . GT . 0 ) THEN
                 JKVV = JE( 1 , KKET )
                 ELSE
                 JKVV = JE( 2 , KKET )
                 END IF
                 JVDELT = JVDELT + 1
                 IVDELT( JVDELT ) = JKVV
                 DO 210 KK = 2 , IJTRIG
                 IICOLR( KK - 1 ) = IICOLR( KK )
210              CONTINUE
                 IJTRIG = IJTRIG - 2
                 IETRIG = IETRIG + 1
                 IECRSS( IETRIG ) = KKET
                 GO TO 200
                 END IF
C
170              CONTINUE
C
                 INDCTR = 2
C                 IF( XYLONG / XYSHRT . GT . 10. . AND . JLOOP . EQ . 0 ) RETURN
C
                 ELSE IF( IKKE . EQ . 1 ) THEN
C
C        BEGINING THE DELETION PROCESS IF KSD HAS AN EDGE ON THE BOUNDARY
C        THE FIRST LOOP IS AROUND VERTEX KKV2 .
C
                 IVV = KKV2
                 IE = IEIN
                 IVIN = JE( 2 , IE )
                 XXYYIB = XE( 1 , IE ) + XE( 1 , IKKE1 )
                 IV1 = JE( 1 , IE )
                 IF( IV1 . EQ . IVV ) THEN
                 ISI = JE( 3 , IE )
                 ELSE
                 ISI = JE( 4 , IE )
                 END IF
                 IS = ISI
C
220              CONTINUE
C
                 ITRIG = ITRIG + 1
                 ISCRSS( ITRIG ) = IS
C
                 IETRIG = IETRIG + 1
                 IECRSS( IETRIG ) = IE
C
                 IF(
         .            HYDFLX( IS , 4 ) . GT . FLUXPP . OR .
```

```
     .        HYDFLX( IS , 2 ) . GT . FLUXUU . OR .
     .        HYDFLX( IS , 1 ) . GT . FLUXRR . OR .
     .        KSDELT( IS ) . GT . NIDUMP . OR .
     .        XS( 3 , IS ) . GT . AREVGG ) THEN
           INDCTR = 3
           RETURN
           END IF
C
           DO 230 IR = 1 , 3
           JR = MOD( IR , 3 ) + 1
           IEA = IABS( JS( JR + 3 , IS ) )
           IF( IEA . EQ . IE ) THEN
           IIR = MOD( JR , 3 ) + 4
           IEI = JS( IIR , IS )
           IEIB = IABS( IEI )
           XEIEB = XE( 1 , IEIB )
           XYLNGT = XYLNGT + XEIEB
           IF( XYLONG . LT . XEIEB ) XYLONG = XEIEB
           IF( XYSHRT . GT . XEIEB ) XYSHRT = XEIEB
           IJTRIG = IJTRIG + 1
           IICOLR( IJTRIG ) = IEI
           JJR = MOD( JR + 1 , 3 ) + 4
           IER = IABS( JS( JJR , IS ) )
C
           IV1 = JE( 1 , IER )
           IF( IV1 . EQ . IVV ) THEN
           ISR = JE( 3 , IER )
           ELSE
           ISR = JE( 4 , IER )
           END IF
           END IF
C
230        CONTINUE
C
           IF( IER . NE . IKKE1 ) THEN
           IS = ISR
           IE = IER
           GO TO 220
           END IF
C
           IETRIG = IETRIG + 1
           IECRSS( IETRIG ) = IKKE1
           IJTRIG = IJTRIG - 2
C
C     FIRST LOOP SUROUNDING KKV2 IS DONE, SECOND LOOP OVER KKV3 START .
C
           IVV = KKV3
           IE = IABS( IICOLR( IJTRIG + 1 ) )
           IV1 = JE( 1 , IE )
           IF( IV1 . EQ . IVV ) THEN
           ISI = JE( 3 , IE )
           ELSE
           ISI = JE( 4 , IE )
           END IF
           IS = ISI
C
           ILOOP = 0
240        CONTINUE
           JDOUBL = IABS( IICOLR( IJTRIG ) )
C
           ITRIG = ITRIG + 1
           ISCRSS( ITRIG ) = IS
           IETRIG = IETRIG + 1
           IECRSS( IETRIG ) = IE
C
           IF(
     .        HYDFLX( IS , 4 ) . GT . FLUXPP . OR .
     .        HYDFLX( IS , 2 ) . GT . FLUXUU . OR .
     .        HYDFLX( IS , 1 ) . GT . FLUXRR . OR .
     .        KSDELT( IS ) . GT . NIDUMP . OR .
     .        XS( 3 , IS ) . GT . AREVGG ) THEN
           INDCTR = 3
           RETURN
           END IF
```

```
C
      DO 250 IR = 1 , 3
      JR = MOD( IR , 3 ) + 1
      IEA = IABS( JS( JR + 3 , IS ) )
      IF( IEA . EQ . IE ) THEN
      IIR = MOD( JR , 3 ) + 4
      IEI = JS( IIR , IS )
      IEIB = IABS( IEI )
      XEIEB = XE( 1 , IEIB )
      XYLNGT = XYLNGT + XEIEB
      IF( XYLONG . LT . XEIEB ) XYLONG = XEIEB
      IF( XYSHRT . GT . XEIEB ) XYSHRT = XEIEB
      ILOOP = ILOOP + 1
      IF( ILOOP . EQ . 1 . AND . JDOUBL . EQ . IEIB ) THEN
      JLOOP = 1
      IETRIG = IETRIG + 1
      IECRSS( IETRIG ) = JDOUBL
      IJTRIG = IJTRIG - 1
      IF( IEI . GT . 0 ) THEN
      JKVV = JE( 1 , IEIB )
      ELSE
      JKVV = JE( 2 , IEIB )
      END IF
      JVDELT = JVDELT + 1
      IVDELT( JVDELT ) = JKVV
      ILOOP = 0
      ELSE
      IJTRIG = IJTRIG + 1
      IICOLR( IJTRIG ) = IEI
      END IF
      JJR = MOD( JR + 1 , 3 ) + 4
      IER = IABS( JS( JJR , IS ) )
C
      IV1 = JE( 1 , IER )
      IF( IV1 . EQ . IVV ) THEN
      ISR = JE( 3 , IER )
      ELSE
      ISR = JE( 4 , IER )
      END IF
      END IF
C
250   CONTINUE
C
      IF( IER . NE . IKKE3 ) THEN
      IS = ISR
      IE = IER
      GO TO 240
      END IF
C
      IETRIG = IETRIG + 1
      IECRSS( IETRIG ) = IKKE3
      IJTRIG = IJTRIG - 1
C
C   SECOND LOOP SUROUNDING KKV3 IS DONE, THIRD LOOP OVER KKV1 START .
C
      IVV = KKV1
      IE = IABS( IICOLR( IJTRIG + 1 ) )
      IF( JE( 5 , IE ) . NE . 0 ) THEN
      IER = IE
      GO TO 260
      END IF
      IV1 = JE( 1 , IE )
      IF( IV1 . EQ . IVV ) THEN
      ISI = JE( 3 , IE )
      ELSE
      ISI = JE( 4 , IE )
      END IF
      IS = ISI
      ISI = 0
C
      ILOOP = 0
270   CONTINUE
      KDOUBL = IABS( IICOLR( IJTRIG ) )
C
```

```
                ITRIG = ITRIG + 1
                ISCRSS( ITRIG ) = IS
                IETRIG = IETRIG + 1
                IECRSS( IETRIG ) = IE
C
                IF(
     .             HYDFLX( IS , 4 ) . GT . FLUXPP . OR .
     .             HYDFLX( IS , 2 ) . GT . FLUXUU . OR .
     .             HYDFLX( IS , 1 ) . GT . FLUXRR . OR .
     .             KSDELT( IS ) . GT . NIDUMP . OR .
     .             XS( 3 , IS ) . GT . AREVGG ) THEN
                INDCTR = 3
                RETURN
                END IF
C
                DO 280 IR = 1 , 3
                JR = MOD( IR , 3 ) + 1
                IEA = IABS( JS( JR + 3 , IS ) )
                IF( IEA . EQ . IE ) THEN
                IIR = MOD( JR , 3 ) + 4
                IEI = JS( IIR , IS )
                IEIB = IABS( IEI )
                XEIEB = XE( 1 , IEIB )
                XYLNGT = XYLNGT + XEIEB
                IF( XYLONG . LT . XEIEB ) XYLONG = XEIEB
                IF( XYSHRT . GT . XEIEB ) XYSHRT = XEIEB
                ILOOP = ILOOP + 1
                IF( ILOOP . EQ . 1 . AND . KDOUBL . EQ . IEIB ) THEN
                JLOOP = 2
                IETRIG = IETRIG + 1
                IECRSS( IETRIG ) = KDOUBL
                IJTRIG = IJTRIG - 1
                IF( IEI . GT . 0 ) THEN
                JKVV = JE( 1 , IEIB )
                ELSE
                JKVV = JE( 2 , IEIB )
                END IF
                JVDELT = JVDELT + 1
                IVDELT( JVDELT ) = JKVV
                ILOOP = 0
                ELSE
                IJTRIG = IJTRIG + 1
                IICOLR( IJTRIG ) = IEI
                END IF
                JJR = MOD( JR + 1 , 3 ) + 4
                IER = IABS( JS( JJR , IS ) )
C
                IV1 = JE( 1 , IER )
                IF( IV1 . EQ . IVV ) THEN
                ISR = JE( 3 , IER )
                ELSE
                ISR = JE( 4 , IER )
                END IF
                END IF
C
280             CONTINUE
C
                IF( ISR . NE . ISI ) THEN
                IS = ISR
                IE = IER
                GO TO 270
                END IF
C
260             CONTINUE
C
                IETRIG = IETRIG + 1
                IECRSS( IETRIG ) = IER
C
                ITYPE = JE( 5 , IER )
C
                XEIEB = XE( 1 , IER )
                XEIEB = XXYYIB + XEIEB
                XYLNGT = XYLNGT + XEIEB
                IF( XYLONG . LT . XEIEB ) XYLONG = XEIEB
```

```
      IF( XYSHRT . GT . XEIEB ) XYSHRT = XEIEB
C
      INDCTR = 2
C      IF( XYLONG / XYSHRT . GT . 10. . AND . JLOOP . EQ . 0 ) RETURN
C
      IV1 = IVIN
      IE1 = IICOLR( IJTRIG )
      IF( IE1 . GT . 0 ) THEN
      IV2 = JE( 2 , IE1 )
      ELSE
      IV2 = JE( 1 , - IE1 )
      END IF
C
      NEC = IECRSS( IETRIG )
      IETRIG = IETRIG - 1
C
      JV( 2 , IV2 ) = - NEC
      JE( 1 , NEC ) = IV2
      JE( 2 , NEC ) = IV1
      JE( 4 , NEC ) = 0
      JE( 5 , NEC ) = ITYPE
C
      IJTRIG = IJTRIG + 1
      IICOLR( IJTRIG ) = NEC
C
      ELSE IF( IKKE . EQ . 2 ) THEN
C
C   BEGINING THE DELETION PROCESS IF KSD HAS A VERTEX ON THE BOUNDARY
C   THE FIRST LOOP IS AROUND VERTEX KKV1 .
C
      IVV = KKV1
      IE = IEIN
      IVIN = JE( 2 , IE )
      XXYYIB = XE( 1 , IE )
      IV1 = JE( 1 , IE )
      IF( IV1 . EQ . IVV ) THEN
      ISI = JE( 3 , IE )
      ELSE
      ISI = JE( 4 , IE )
      END IF
      IS = ISI
C
  290 CONTINUE
C
      ITRIG = ITRIG + 1
      ISCRSS( ITRIG ) = IS
C
      IETRIG = IETRIG + 1
      IECRSS( IETRIG ) = IE
C
      IF(
     .    HYDFLX( IS , 4 ) . GT . FLUXPP . OR .
     .    HYDFLX( IS , 2 ) . GT . FLUXUU . OR .
     .    HYDFLX( IS , 1 ) . GT . FLUXRR . OR .
     .    KSDELT( IS ) . GT . NIDUMP . OR .
     .    XS( 3 , IS ) . GT . AREVGG ) THEN
      INDCTR = 3
      RETURN
      END IF
C
      DO 300 IR = 1 , 3
      JR = MOD( IR , 3 ) + 1
      IEA = IABS( JS( JR + 3 , IS ) )
      IF( IEA . EQ . IE ) THEN
      IIR = MOD( JR , 3 ) + 4
      IEI = JS( IIR , IS )
      IEIB = IABS( IEI )
      XEIEB = XE( 1 , IEIB )
      XYLNGT = XYLNGT + XEIEB
      IF( XYLONG . LT . XEIEB ) XYLONG = XEIEB
      IF( XYSHRT . GT . XEIEB ) XYSHRT = XEIEB
      IJTRIG = IJTRIG + 1
      IICOLR( IJTRIG ) = IEI
      JJR = MOD( JR + 1 , 3 ) + 4
```

```
            IER = IABS( JS( JJR , IS ) )
      C
            IV1 = JE( 1 , IER )
            IF( IV1 . EQ . IVV ) THEN
            ISR = JE( 3 , IER )
            ELSE
            ISR = JE( 4 , IER )
            END IF
            END IF
      C
      300   CONTINUE
      C
            IF( IER . NE . IKKE3 ) THEN
            IS = ISR
            IE = IER
            GO TO 290
            END IF
            IJTRIG = IJTRIG - 2
      C
      C  FIRST LOOP SUROUNDING KKV1 IS DONE, SECOND LOOP OVER KKV2 START .
      C
            IVV = KKV2
            IE = IABS( IICOLR( IJTRIG + 1 ) )
            IV1 = JE( 1 , IE )
            IF( IV1 . EQ . IVV ) THEN
            ISI = JE( 3 , IE )
            ELSE
            ISI = JE( 4 , IE )
            END IF
            IS = ISI
      C
            ILOOP = 0
      310   CONTINUE
            IDOUBL = IABS( IICOLR( IJTRIG ) )
      C
            ITRIG = ITRIG + 1
            ISCRSS( ITRIG ) = IS
      C
            IETRIG = IETRIG + 1
            IECRSS( IETRIG ) = IE
      C
            IF(
     .          HYDFLX( IS , 4 ) . GT . FLUXPP . OR .
     .          HYDFLX( IS , 2 ) . GT . FLUXUU . OR .
     .          HYDFLX( IS , 1 ) . GT . FLUXRR . OR .
     .          KSDELT( IS ) . GT . NIDUMP . OR .
     .          XS( 3 , IS ) . GT . AREVGG ) THEN
            INDCTR = 3
            RETURN
            END IF
      C
            DO 320 IR = 1 , 3
            JR = MOD( IR , 3 ) + 1
            IEA = IABS( JS( JR + 3 , IS ) )
            IF( IEA . EQ . IE ) THEN
            IIR = MOD( JR , 3 ) + 4
            IEI = JS( IIR , IS )
            IEIB = IABS( IEI )
            XEIEB = XE( 1 , IEIB )
            XYLNGT = XYLNGT + XEIEB
            IF( XYLONG . LT . XEIEB ) XYLONG = XEIEB
            IF( XYSHRT . GT . XEIEB ) XYSHRT = XEIEB
            ILOOP = ILOOP + 1
            IF( ILOOP . EQ . 1 . AND . IDOUBL . EQ . IEIB ) THEN
            JLOOP = 1
            IETRIG = IETRIG + 1
            IECRSS( IETRIG ) = IDOUBL
            IJTRIG = IJTRIG - 1
            IF( IEI . GT . 0 ) THEN
            JKVV = JE( 1 , IEIB )
            ELSE
            JKVV = JE( 2 , IEIB )
            END IF
            JVDELT = JVDELT + 1
```

```
                IVDELT( JVDELT ) = JKVV
                ILOOP = 0
                ELSE
                IJTRIG = IJTRIG + 1
                IICOLR( IJTRIG ) = IEI
                END IF
                JJR = MOD( JR + 1 , 3 ) + 4
                IER = IABS( JS( JJR , IS ) )
C
                IV1 = JE( 1 , IER )
                IF( IV1 . EQ . IVV ) THEN
                ISR = JE( 3 , IER )
                ELSE
                ISR = JE( 4 , IER )
                END IF
                END IF
C
320             CONTINUE
C
                IF( IER . NE . IKKE2 ) THEN
                IS = ISR
                IE = IER
                GO TO 310
                END IF
C
                IJTRIG = IJTRIG - 1
C
C     SECOND LOOP SUROUNDING KKV2 IS DONE, THIRD LOOP OVER KKV3 START .
C
                IVV = KKV3
                IE = IABS( IICOLR( IJTRIG + 1 ) )
                IV1 = JE( 1 , IE )
                IF( IV1 . EQ . IVV ) THEN
                ISI = JE( 3 , IE )
                ELSE
                ISI = JE( 4 , IE )
                END IF
                IS = ISI
C
                ILOOP = 0
330             CONTINUE
                KDOUBL = IABS( IICOLR( IJTRIG ) )
C
                ITRIG = ITRIG + 1
                ISCRSS( ITRIG ) = IS
                IETRIG = IETRIG + 1
                IECRSS( IETRIG ) = IE
C
                IF(
     .              HYDFLX( IS , 4 ) . GT . FLUXPP . OR .
     .              HYDFLX( IS , 2 ) . GT . FLUXUU . OR .
     .              HYDFLX( IS , 1 ) . GT . FLUXRR . OR .
     .              KSDELT( IS ) . GT . NIDUMP . OR .
     .              XS( 3 , IS ) . GT . AREVGG ) THEN
                INDCTR = 3
                RETURN
                END IF
C
                DO 340 IR = 1 , 3
                JR = MOD( IR , 3 ) + 1
                IEA = IABS( JS( JR + 3 , IS ) )
                IF( IEA . EQ . IE ) THEN
                IIR = MOD( JR , 3 ) + 4
                IEI = JS( IIR , IS )
                IEIB = IABS( IEI )
                XEIEB = XE( 1 , IEIB )
                XYLNGT = XYLNGT + XEIEB
                IF( XYLONG . LT . XEIEB ) XYLONG = XEIEB
                IF( XYSHRT . GT . XEIEB ) XYSHRT = XEIEB
                ILOOP = ILOOP + 1
                IF( ILOOP . EQ . 1 . AND . KDOUBL . EQ . IEIB ) THEN
                JLOOP = 2
                IETRIG = IETRIG + 1
                IECRSS( IETRIG ) = KDOUBL
```

```
            IJTRIG = IJTRIG - 1
            IF( IEI . GT . 0 ) THEN
            JKVV = JE( 1 , IEIB )
            ELSE
            JKVV = JE( 2 , IEIB )
            END IF
            JVDELT = JVDELT + 1
            IVDELT( JVDELT ) = JKVV
            ILOOP = 0
            ELSE
            IJTRIG = IJTRIG + 1
            IICOLR( IJTRIG ) = IEI
            END IF
            JJR = MOD( JR + 1 , 3 ) + 4
            IER = IABS( JS( JJR , IS ) )
C
            IV1 = JE( 1 , IER )
            IF( IV1 . EQ . IVV ) THEN
            ISR = JE( 3 , IER )
            ELSE
            ISR = JE( 4 , IER )
            END IF
            END IF
C
  340       CONTINUE
C
            IF( IER . NE . IKKE3 ) THEN
            IS = ISR
            IE = IER
            GO TO 330
            END IF
C
            IETRIG = IETRIG + 1
            IECRSS( IETRIG ) = IKKE3
            IETRIG = IETRIG + 1
            IECRSS( IETRIG ) = IKKE2
C
            IJTRIG = IJTRIG - 1
C
C      THIRD LOOP SUROUNDING KKV3 IS DONE, FOURTH LOOP OVER KKV1 START .
C
            IVV = KKV1
            IE = IABS( IICOLR( IJTRIG + 1 ) )
            IF( JE( 5 , IE ) . NE . 0 ) THEN
            IER = IE
            GO TO 350
            END IF
            IV1 = JE( 1 , IE )
            IF( IV1 . EQ . IVV ) THEN
            ISI = JE( 3 , IE )
            ELSE
            ISI = JE( 4 , IE )
            END IF
            IS = ISI
            ISI = 0
C
            ILOOP = 0
  360       CONTINUE
            JDOUBL = IABS( IICOLR( IJTRIG ) )
C
            ITRIG = ITRIG + 1
            ISCRSS( ITRIG ) = IS
            IETRIG = IETRIG + 1
            IECRSS( IETRIG ) = IE
C
            IF(
     .          HYDFLX( IS , 4 ) . GT . FLUXPP . OR .
     .          HYDFLX( IS , 2 ) . GT . FLUXUU . OR .
     .          HYDFLX( IS , 1 ) . GT . FLUXRR . OR .
     .          KSDELT( IS ) . GT . NIDUMP . OR .
     .          XS( 3 , IS ) . GT . AREVGG ) THEN
            INDCTR = 3
            RETURN
            END IF
```

```
C
          DO 370 IR = 1 , 3
          JR = MOD( IR , 3 ) + 1
          IEA = IABS( JS( JR + 3 , IS ) )
          IF( IEA . EQ . IE ) THEN
          IIR = MOD( JR , 3 ) + 4
          IEI = JS( IIR , IS )
          IEIB = IABS( IEI )
          XEIEB = XE( 1 , IEIB )
          XYLNGT = XYLNGT + XEIEB
          IF( XYLONG . LT . XEIEB ) XYLONG = XEIEB
          IF( XYSHRT . GT . XEIEB ) XYSHRT = XEIEB
          ILOOP = ILOOP + 1
          IF( ILOOP . EQ . 1 . AND . JDOUBL . EQ . IEIB ) THEN
          JLOOP = 3
          IETRIG = IETRIG + 1
          IECRSS( IETRIG ) = JDOUBL
          IJTRIG = IJTRIG - 1
          IF( IEI . GT . 0 ) THEN
          JKVV = JE( 1 , IEIB )
          ELSE
          JKVV = JE( 2 , IEIB )
          END IF
          JVDELT = JVDELT + 1
          IVDELT( JVDELT ) = JKVV
          ILOOP = 0
          ELSE
          IJTRIG = IJTRIG + 1
          IICOLR( IJTRIG ) = IEI
          END IF
          JJR = MOD( JR + 1 , 3 ) + 4
          IER = IABS( JS( JJR , IS ) )
C
          IV1 = JE( 1 , IER )
          IF( IV1 . EQ . IVV ) THEN
          ISR = JE( 3 , IER )
          ELSE
          ISR = JE( 4 , IER )
          END IF
          END IF
C
370       CONTINUE
C
          IF( ISR . NE . ISI ) THEN
          IS = ISR
          IE = IER
          GO TO 360
          END IF
C
350       CONTINUE
C
          IETRIG = IETRIG + 1
          IECRSS( IETRIG ) = IER
C
          ITYPE = JE( 5 , IER )
C
          XEIEB = XE( 1 , IER )
          XEIEB = XXYYIB + XEIEB
          XYLNGT = XYLNGT + XEIEB
          IF( XYLONG . LT . XEIEB ) XYLONG = XEIEB
          IF( XYSHRT . GT . XEIEB ) XYSHRT = XEIEB
C
          INDCTR = 2
C          IF( XYLONG / XYSHRT . GT . 10. . AND . JLOOP . EQ . 0 ) RETURN
C
          IV1 = IVIN
          IE1 = IICOLR( IJTRIG )
          IF( IE1 . GT . 0 ) THEN
          IV2 = JE( 2 , IE1 )
          ELSE
          IV2 = JE( 1 , - IE1 )
          END IF
C
          NEC = IECRSS( IETRIG )
```

```
            IETRIG = IETRIG - 1
C
            JV( 2 , IV2 ) = - NEC
            JE( 1 , NEC ) = IV2
            JE( 2 , NEC ) = IV1
            JE( 4 , NEC ) = 0
            JE( 5 , NEC ) = ITYPE
C
            IJTRIG = IJTRIG + 1
            IICOLR( IJTRIG ) = NEC
C
            ELSE IF( IKKE . EQ . 3 ) THEN
C
C     BEGINING THE DELETION PROCESS IF KSD HAS TWO VERTECIS ON THE BOUNDARY
C     BUT THE EDGE THAT CONNECT THEM IS IN THE COMPUTATIONAL DOMAIN.
C     THE FIRST LOOP IS AROUND VERTEX KKV1 .
C
            IVV = KKV1
            IE = IEIN1
            XXYYIB = XE( 1 , IE )
            IV1 = JE( 1 , IE )
            IVIN1 = JE( 2 , IE )
            IF( IV1 . EQ . IVV ) THEN
            ISI = JE( 3 , IE )
            ELSE
            ISI = JE( 4 , IE )
            END IF
            IS = ISI
C
  380       CONTINUE
C
            ITRIG = ITRIG + 1
            ISCRSS( ITRIG ) = IS
            IETRIG = IETRIG + 1
            IECRSS( IETRIG ) = IE
C
            IF(
     .          HYDFLX( IS , 4 ) . GT . FLUXPP . OR .
     .          HYDFLX( IS , 2 ) . GT . FLUXUU . OR .
     .          HYDFLX( IS , 1 ) . GT . FLUXRR . OR .
     .          KSDELT( IS ) . GT . NIDUMP . OR .
     .          XS( 3 , IS ) . GT . AREVGG ) THEN
            INDCTR = 3
            RETURN
            END IF
C
            DO 390 IR = 1 , 3
            JR = MOD( IR , 3 ) + 1
            IEA = IABS( JS( JR + 3 , IS ) )
            IF( IEA . EQ . IE ) THEN
            IIR = MOD( JR , 3 ) + 4
            IEI = JS( IIR , IS )
            IEIB = IABS( IEI )
            XEIEB = XE( 1 , IEIB )
            XYLNGT = XYLNGT + XEIEB
            IF( XYLONG . LT . XEIEB ) XYLONG = XEIEB
            IF( XYSHRT . GT . XEIEB ) XYSHRT = XEIEB
            IJTRIG = IJTRIG + 1
            IICOLR( IJTRIG ) = IEI
            JJR = MOD( JR + 1 , 3 ) + 4
            IER = IABS( JS( JJR , IS ) )
C
            IV1 = JE( 1 , IER )
            IF( IV1 . EQ . IVV ) THEN
            ISR = JE( 3 , IER )
            ELSE
            ISR = JE( 4 , IER )
            END IF
            END IF
C
  390       CONTINUE
C
            IF( IER . NE . IKKE3 ) THEN
            IS = ISR
```

```
                IE = IER
                GO TO 380
                END IF
      C
      C    FIRST LOOP SUROUNDING KKV1 IS DONE, SECOND LOOP OVER KKV2 START .
      C
                IJTRIG = IJTRIG - 1
      400       CONTINUE
      C
                IEJK = IICOLR( IJTRIG )
                IF( IEJK . GT . 0 ) THEN
                IVIEJK = JE( 1 , IEJK )
                IJEJK5 = JE( 5 , IEJK )
                ELSE
                IVIEJK = JE( 2 , -IEJK )
                IJEJK5 = JE( 5 , -IEJK )
                END IF
      C
                IF( IJEJK5 . EQ . 0 ) THEN
                JLOOP = 1
      C
      C    INTERMEDIATE LOOP START .
      C
                IEJKI = IABS( IICOLR( IJTRIG - 1 ) )
                IEJK2 = IABS( IEJK )
                IETRIG = IETRIG + 1
                IECRSS( IETRIG ) = IEJK2
                IJTRIG = IJTRIG - 2
                IVV = IVIEJK
                JVDELT = JVDELT + 1
                IVDELT( JVDELT ) = IVV
                IE = IEJKI
                IV1 = JE( 1 , IE )
                IF( IV1 . EQ . IVV ) THEN
                ISI = JE( 3 , IE )
                ELSE
                ISI = JE( 4 , IE )
                END IF
                IS = ISI
                IET = IEJK2
      C
      410       CONTINUE
      C
                ITRIG = ITRIG + 1
                ISCRSS( ITRIG ) = IS
      C
                IETRIG = IETRIG + 1
                IECRSS( IETRIG ) = IE
      C
                IF(
           .        HYDFLX( IS , 4 ) . GT . FLUXPP . OR .
           .        HYDFLX( IS , 2 ) . GT . FLUXUU . OR .
           .        HYDFLX( IS , 1 ) . GT . FLUXRR . OR .
           .        KSDELT( IS ) . GT . NIDUMP . OR .
           .        XS( 3 , IS ) . GT . AREVGG ) THEN
                INDCTR = 3
                RETURN
                END IF
      C
                DO 420 IR = 1 , 3
                JR = MOD( IR , 3 ) + 1
                IEA = IABS( JS( JR + 3 , IS ) )
                IF( IEA . EQ . IE ) THEN
                IIR = MOD( JR , 3 ) + 4
                IEI = JS( IIR , IS )
                IEIB = IABS( IEI )
                XEIEB = XE( 1 , IEIB )
                XYLNGT = XYLNGT + XEIEB
                IF( XYLONG . LT . XEIEB ) XYLONG = XEIEB
                IF( XYSHRT . GT . XEIEB ) XYSHRT = XEIEB
                IIKK = IABS( IICOLR( IJTRIG ) )
                IF( IIKK . EQ . IEIB ) THEN
                JLOOP = 2
                IETRIG = IETRIG + 1
```

```
                  IECRSS( IETRIG ) = IEIB
                  IJTRIG = IJTRIG - 1
                  IF( IEI . GT . 0 ) THEN
                  JKVV = JE( 1 , IEIB )
                  ELSE
                  JKVV = JE( 2 , IEIB )
                  END IF
                  JVDELT = JVDELT + 1
                  IVDELT( JVDELT ) = JKVV
                  ELSE
                  IJTRIG = IJTRIG + 1
                  IICOLR( IJTRIG ) = IEI
                  END IF
                  JJR = MOD( JR + 1 , 3 ) + 4
                  IER = IABS( JS( JJR , IS ) )
C
                  IV1 = JE( 1 , IER )
                  IF( IV1 . EQ . IVV ) THEN
                  ISR = JE( 3 , IER )
                  ELSE
                  ISR = JE( 4 , IER )
                  END IF
                  END IF
C
420               CONTINUE
C
                  IF( IER . NE . IET ) THEN
                  IS = ISR
                  IE = IER
                  GO TO 410
                  END IF
C
                  GO TO 400
                  END IF
C
C      INTERMEDIATE LOOP IS DONE, SECOND LOOP OVER KKV2 START .
C
                  IVV = KKV2
                  IE = IEIN2
                  IVIN2 = JE( 2 , IE )
                  IEJKK = IICOLR( IJTRIG )
                  IV1 = JE( 1 , IE )
                  IF( IV1 . EQ . IVV ) THEN
                  ISI = JE( 3 , IE )
                  ELSE
                  ISI = JE( 4 , IE )
                  END IF
                  IS = ISI
C
430               CONTINUE
C
                  ITRIG = ITRIG + 1
                  ISCRSS( ITRIG ) = IS
C
                  IETRIG = IETRIG + 1
                  IECRSS( IETRIG ) = IE
C
                  IF(
     .               HYDFLX( IS , 4 ) . GT . FLUXPP . OR .
     .               HYDFLX( IS , 2 ) . GT . FLUXUU . OR .
     .               HYDFLX( IS , 1 ) . GT . FLUXRR . OR .
     .               KSDELT( IS ) . GT . NIDUMP . OR .
     .               XS( 3 , IS ) . GT . AREVGG ) THEN
                  INDCTR = 3
                  RETURN
                  END IF
C
                  DO 440 IR = 1 , 3
                  JR = MOD( IR , 3 ) + 1
                  IEA = IABS( JS( JR + 3 , IS ) )
                  IF( IEA . EQ . IE ) THEN
                  IIR = MOD( JR , 3 ) + 4
                  IEI = JS( IIR , IS )
                  IEIB = IABS( IEI )
```

```
            XEIEB = XE( 1 , IEIB )
            XYLNGT = XYLNGT + XEIEB
            IF( XYLONG . LT . XEIEB ) XYLONG = XEIEB
            IF( XYSHRT . GT . XEIEB ) XYSHRT = XEIEB
            IJTRIG = IJTRIG + 1
            IICOLR( IJTRIG ) = IEI
            JJR = MOD( JR + 1 , 3 ) + 4
            IER = IABS( JS( JJR , IS ) )
C
            IV1 = JE( 1 , IER )
            IF( IV1 . EQ . IVV ) THEN
            ISR = JE( 3 , IER )
            ELSE
            ISR = JE( 4 , IER )
            END IF
            END IF
C
440         CONTINUE
C
            IF( IER . NE . IKKE2 ) THEN
            IS = ISR
            IE = IER
            GO TO 430
            END IF
C
            IJTRIG = IJTRIG - 1
C
C     SECOND LOOP SUROUNDING KKV2 IS DONE, THIRD LOOP OVER KKV3 START .
C
            IVV = KKV3
            IE = IABS( IICOLR( IJTRIG + 1 ) )
            IV1 = JE( 1 , IE )
            IF( IV1 . EQ . IVV ) THEN
            ISI = JE( 3 , IE )
            ELSE
            ISI = JE( 4 , IE )
            END IF
            IS = ISI
C
            ILOOP = 0
450         CONTINUE
            IDOUBL = IABS( IICOLR( IJTRIG ) )
C
            ITRIG = ITRIG + 1
            ISCRSS( ITRIG ) = IS
            IETRIG = IETRIG + 1
            IECRSS( IETRIG ) = IE
C
            IF(
     .          HYDFLX( IS , 4 ) . GT . FLUXPP . OR .
     .          HYDFLX( IS , 2 ) . GT . FLUXUU . OR .
     .          HYDFLX( IS , 1 ) . GT . FLUXRR . OR .
     .          KSDELT( IS ) . GT . NIDUMP . OR .
     .          XS( 3 , IS ) . GT . AREVGG ) THEN
            INDCTR = 3
            RETURN
            END IF
C
            DO 460 IR = 1 , 3
            JR = MOD( IR , 3 ) + 1
            IEA = IABS( JS( JR + 3 , IS ) )
            IF( IEA . EQ . IE ) THEN
            IIR = MOD( JR , 3 ) + 4
            IEI = JS( IIR , IS )
            IEIB = IABS( IEI )
            XEIEB = XE( 1 , IEIB )
            XYLNGT = XYLNGT + XEIEB
            IF( XYLONG . LT . XEIEB ) XYLONG = XEIEB
            IF( XYSHRT . GT . XEIEB ) XYSHRT = XEIEB
            ILOOP = ILOOP + 1
            IF( ILOOP . EQ . 1 . AND . IDOUBL . EQ . IEIB ) THEN
            JLOOP = 3
            IETRIG = IETRIG + 1
            IECRSS( IETRIG ) = IDOUBL
```

```
            IJTRIG = IJTRIG - 1
            IF( IEI . GT . 0 ) THEN
            JKVV = JE( 1 , IEIB )
            ELSE
            JKVV = JE( 2 , IEIB )
            END IF
            JVDELT = JVDELT + 1
            IVDELT( JVDELT ) = JKVV
            ILOOP = 0
            ELSE
            IJTRIG = IJTRIG + 1
            IICOLR( IJTRIG ) = IEI
            END IF
            JJR = MOD( JR + 1 , 3 ) + 4
            IER = IABS( JS( JJR , IS ) )
C
            IV1 = JE( 1 , IER )
            IF( IV1 . EQ . IVV ) THEN
            ISR = JE( 3 , IER )
            ELSE
            ISR = JE( 4 , IER )
            END IF
            END IF
C
460         CONTINUE
C
            IF( IER . NE . IKKE3 ) THEN
            IS = ISR
            IE = IER
            GO TO 450
            END IF
C
            IETRIG = IETRIG + 1
            IECRSS( IETRIG ) = IKKE3
            IETRIG = IETRIG + 1
            IECRSS( IETRIG ) = IKKE2
C
            IJTRIG = IJTRIG - 1
C
C    THIRD LOOP SUROUNDING KKV3 IS DONE, FOURTH LOOP OVER KKV1 START .
C
            IVV = KKV1
            IE = IABS( IICOLR( IJTRIG + 1 ) )
            IF( JE( 5 , IE ) . NE . 0 ) THEN
            IER = IE
            GO TO 470
            END IF
            IV1 = JE( 1 , IE )
            IF( IV1 . EQ . IVV ) THEN
            ISI = JE( 3 , IE )
            ELSE
            ISI = JE( 4 , IE )
            END IF
            IS = ISI
            ISI = 0
C
            ILOOP = 0
480         CONTINUE
            JDOUBL = IABS( IICOLR( IJTRIG ) )
C
            ITRIG = ITRIG + 1
            ISCRSS( ITRIG ) = IS
            IETRIG = IETRIG + 1
            IECRSS( IETRIG ) = IE
C
            IF(
     .          HYDFLX( IS , 4 ) . GT . FLUXPP . OR .
     .          HYDFLX( IS , 2 ) . GT . FLUXUU . OR .
     .          HYDFLX( IS , 1 ) . GT . FLUXRR . OR .
     .          KSDELT( IS ) . GT . NIDUMP . OR .
     .          XS( 3 , IS ) . GT . AREVGG ) THEN
            INDCTR = 3
            RETURN
            END IF
```

```
C
      DO 490 IR = 1 , 3
      JR = MOD( IR , 3 ) + 1
      IEA = IABS( JS( JR + 3 , IS ) )
      IF( IEA . EQ . IE ) THEN
      IIR = MOD( JR , 3 ) + 4
      IEI = JS( IIR , IS )
      IEIB = IABS( IEI )
      XEIEB = XE( 1 , IEIB )
      XYLNGT = XYLNGT + XEIEB
      IF( XYLONG . LT . XEIEB ) XYLONG = XEIEB
      IF( XYSHRT . GT . XEIEB ) XYSHRT = XEIEB
      ILOOP = ILOOP + 1
      IF( ILOOP . EQ . 1 . AND . JDOUBL . EQ . IEIB ) THEN
      JLOOP = 4
      IETRIG = IETRIG + 1
      IECRSS( IETRIG ) = JDOUBL
      IJTRIG = IJTRIG - 1
      IF( IEI . GT . 0 ) THEN
      JKVV = JE( 1 , IEIB )
      ELSE
      JKVV = JE( 2 , IEIB )
      END IF
      JVDELT = JVDELT + 1
      IVDELT( JVDELT ) = JKVV
      ILOOP = 0
      ELSE
      IJTRIG = IJTRIG + 1
      IICOLR( IJTRIG ) = IEI
      END IF
      JJR = MOD( JR + 1 , 3 ) + 4
      IER = IABS( JS( JJR , IS ) )
C
      IV1 = JE( 1 , IER )
      IF( IV1 . EQ . IVV ) THEN
      ISR = JE( 3 , IER )
      ELSE
      ISR = JE( 4 , IER )
      END IF
      END IF
C
490   CONTINUE
C
      IF( ISR . NE . ISI ) THEN
      IS = ISR
      IE = IER
      GO TO 480
      END IF
C
470   CONTINUE
C
      IETRIG = IETRIG + 1
      IECRSS( IETRIG ) = IER
C
      ITYPE = JE( 5 , IER )
C
      XEIEB = XE( 1 , IER )
      XEIEB = XXYYIB + XEIEB
      XYLNGT = XYLNGT + XEIEB
      IF( XYLONG . LT . XEIEB ) XYLONG = XEIEB
      IF( XYSHRT . GT . XEIEB ) XYSHRT = XEIEB
C
      INDCTR = 2
C      IF( XYLONG / XYSHRT . GT . 10. . AND . JLOOP . EQ . 0 ) RETURN
C
      JE( 2 , IEJKK ) = IVIN2
C
      IV1 = IVIN1
      IE1 = IICOLR( IJTRIG )
      IF( IE1 . GT . 0 ) THEN
      IV2 = JE( 2 , IE1 )
      ELSE
      IV2 = JE( 1 , - IE1 )
      END IF
```

```
C
              NEC = IECRSS( IETRIG )
              IETRIG = IETRIG - 1
C
              JV( 2 , IV2 ) = - NEC
              JE( 1 , NEC ) = IV2
              JE( 2 , NEC ) = IV1
              JE( 4 , NEC ) = 0
              JE( 5 , NEC ) = ITYPE
C
              IJTRIG = IJTRIG + 1
              IICOLR( IJTRIG ) = NEC
C
              ELSE IF( IKKE . EQ . 4 ) THEN
                print*,'ikke=4',ksd,ikke
C
C     BEGINING THE DELETION PROCESS IF KSD HAS AN EDGE ON THE BOUNDARY
C     AND THE THIRD VERTEX IS OLSO ON THE BOUNDARY.
C     THE FIRST LOOP IS AROUND VERTEX KKV2 .
C
              IVV = KKV2
              IE = IEIN1
              XXYYIB = XE( 1 , IE ) + XE( 1 , IKKE1 )
              IV1 = JE( 1 , IE )
              IVIN1 = JE( 2 , IE )
              IF( IV1 . EQ . IVV ) THEN
              ISI = JE( 3 , IE )
              ELSE
              ISI = JE( 4 , IE )
              END IF
              IS = ISI
C
 500          CONTINUE
C
              ITRIG = ITRIG + 1
              ISCRSS( ITRIG ) = IS
              IETRIG = IETRIG + 1
              IECRSS( IETRIG ) = IE
C
              IF(
     .            HYDFLX( IS , 4 ) . GT . FLUXPP . OR .
     .            HYDFLX( IS , 2 ) . GT . FLUXUU . OR .
     .            HYDFLX( IS , 1 ) . GT . FLUXRR . OR .
     .            KSDELT( IS ) . GT . NIDUMP . OR .
     .            XS( 3 , IS ) . GT . AREVGG ) THEN
              INDCTR = 3
              RETURN
              END IF
C
              DO 510 IR = 1 , 3
              JR = MOD( IR , 3 ) + 1
              IEA = IABS( JS( JR + 3 , IS ) )
              IF( IEA . EQ . IE ) THEN
              IIR = MOD( JR , 3 ) + 4
              IEI = JS( IIR , IS )
              IEIB = IABS( IEI )
              XEIEB = XE( 1 , IEIB )
              XYLNGT = XYLNGT + XEIEB
              IF( XYLONG . LT . XEIEB ) XYLONG = XEIEB
              IF( XYSHRT . GT . XEIEB ) XYSHRT = XEIEB
              IJTRIG = IJTRIG + 1
              IICOLR( IJTRIG ) = IEI
              JJR = MOD( JR + 1 , 3 ) + 4
              IER = IABS( JS( JJR , IS ) )
C
              IV1 = JE( 1 , IER )
              IF( IV1 . EQ . IVV ) THEN
              ISR = JE( 3 , IER )
              ELSE
              ISR = JE( 4 , IER )
              END IF
              END IF
C
 510          CONTINUE
```

```
C
      IF( IER . NE . IKKE1 ) THEN
      IS = ISR
      IE = IER
      GO TO 500
      END IF
C
C   FIRST LOOP SUROUNDING KKV2 IS DONE, SECOND LOOP OVER KKV3 START .
C
      IJTRIG = IJTRIG - 1
 520  CONTINUE
C
      IEJK = IICOLR( IJTRIG )
      IF( IEJK . GT . 0 ) THEN
      IVIEJK = JE( 1 , IEJK )
      IJEJK5 = JE( 5 , IEJK )
      ELSE
      IVIEJK = JE( 2 , -IEJK )
      IJEJK5 = JE( 5 , -IEJK )
      END IF
C
      IF( IJEJK5 . EQ . 0 ) THEN
C
C   INTERMEDIATE LOOP START .
C
      JLOOP = 1
      IEJKI = IABS( IICOLR( IJTRIG - 1 ) )
      IEJK2 = IABS( IEJK )
      IETRIG = IETRIG + 1
      IECRSS( IETRIG ) = IEJK2
      IJTRIG = IJTRIG - 2
      IVV = IVIEJK
      JVDELT = JVDELT + 1
      IVDELT( JVDELT ) = IVV
      IE = IEJKI
      IV1 = JE( 1 , IE )
      IF( IV1 . EQ . IVV ) THEN
      ISI = JE( 3 , IE )
      ELSE
      ISI = JE( 4 , IE )
      END IF
      IS = ISI
      IET = IEJK2
C
 530  CONTINUE
C
      ITRIG = ITRIG + 1
      ISCRSS( ITRIG ) = IS
C
      IETRIG = IETRIG + 1
      IECRSS( IETRIG ) = IE
C
      IF(
     .    HYDFLX( IS , 4 ) . GT . FLUXPP . OR .
     .    HYDFLX( IS , 2 ) . GT . FLUXUU . OR .
     .    HYDFLX( IS , 1 ) . GT . FLUXRR . OR .
     .    KSDFLT( IS ) . GT . NIDUMP . OR .
     .    XS( 3 , IS ) . GT . AREVGG ) THEN
      INDCTR = 3
      RETURN
      END IF
C
      DO 540 IR = 1 , 3
      JR = MOD( IR , 3 ) + 1
      IEA = IABS( JS( JR + 3 , IS ) )
      IF( IEA . EQ . IE ) THEN
      IIR = MOD( JR , 3 ) + 4
      IEI = JS( IIR , IS )
      IEIB = IABS( IEI )
      XEIEB = XE( 1 , IEIB )
      XYLNGT = XYLNGT + XEIEB
      IF( XYLONG . LT . XEIEB ) XYLONG = XEIEB
      IF( XYSHRT . GT . XEIEB ) XYSHRT = XEIEB
      IIKK = IABS( IICOLR( IJTRIG ) )
```

```
                  IF( IIKK . EQ . IEIB ) THEN
                  JLOOP = 2
                  IETRIG = IETRIG + 1
                  IECRSS( IETRIG ) = IEIB
                  IJTRIG = IJTRIG - 1
                  IF( IEI . GT . 0 ) THEN
                  JKVV = JE( 1 , IEIB )
                  ELSE
                  JKVV = JE( 2 , IEIB )
                  END IF
                  JVDELT = JVDELT + 1
                  IVDELT( JVDELT ) = JKVV
                  ELSE
                  IJTRIG = IJTRIG + 1
                  IICOLR( IJTRIG ) = IEI
                  END IF
                  JJR = MOD( JR + 1 , 3 ) + 4
                  IER = IABS( JS( JJR , IS ) )
C
                  IV1 = JE( 1 , IER )
                  IF( IV1 . EQ . IVV ) THEN
                  ISR = JE( 3 , IER )
                  ELSE
                  ISR = JE( 4 , IER )
                  END IF
                  END IF
C
  540             CONTINUE
C
                  IF( IER . NE . IET ) THEN
                  IS = ISR
                  IE = IER
                  GO TO 530
                  END IF
C
                  GO TO 520
                  END IF
C
C     INTERMEDIATE LOOP IS DONE, SECOND LOOP OVER KKV3 START .
C
                  IVV = KKV3
                  IE = IEIN2
                  IVIN2 = JE( 2 , IE )
                  IEJKK = IICOLR( IJTRIG )
                  IV1 = JE( 1 , IE )
                  IF( IV1 . EQ . IVV ) THEN
                  ISI = JE( 3 , IE )
                  ELSE
                  ISI = JE( 4 , IE )
                  END IF
                  IS = ISI
C
  550             CONTINUE
C
                  ITRIG = ITRIG + 1
                  ISCRSS( ITRIG ) = IS
C
                  IETRIG = IETRIG + 1
                  IECRSS( IETRIG ) = IE
C
                  IF(
     .                HYDFLX( IS , 4 ) . GT . FLUXPP . OR .
     .                HYDFLX( IS , 2 ) . GT . FLUXUU . OR .
     .                HYDFLX( IS , 1 ) . GT . FLUXRR . OR .
     .                KSDELT( IS ) . GT . NIDUMP . OR .
     .                XS( 3 , IS ) . GT . AREVGG ) THEN
                  INDCTR = 3
                  RETURN
                  END IF
C
                  DO 560 IR = 1 , 3
                  JR = MOD( IR , 3 ) + 1
                  IEA = IABS( JS( JR + 3 , IS ) )
                  IF( IEA . EQ . IE ) THEN
```

```
            IIR = MOD( JR , 3 ) + 4
            IEI = JS( IIR , IS )
            IEIB = IABS( IEI )
            XEIEB = XE( 1 , IEIB )
            XYLNGT = XYLNGT + XEIEB
            IF( XYLONG . LT . XEIEB ) XYLONG = XEIEB
            IF( XYSHRT . GT . XEIEB ) XYSHRT = XEIEB
            IJTRIG = IJTRIG + 1
            IICOLR( IJTRIG ) = IEI
            JJR = MOD( JR + 1 , 3 ) + 4
            IER = IABS( JS( JJR , IS ) )
C
            IV1 = JE( 1 , IER )
            IF( IV1 . EQ . IVV ) THEN
            ISR = JE( 3 , IER )
            ELSE
            ISR = JE( 4 , IER )
            END IF
            END IF
C
560         CONTINUE
C
            IF( IER . NE . IKKE3 ) THEN
            IS = ISR
            IE = IER
            GO TO 550
            END IF
C
            IJTRIG = IJTRIG - 1
            IETRIG = IETRIG + 1
            IECRSS( IETRIG ) = IKKE3
            IETRIG = IETRIG + 1
            IECRSS( IETRIG ) = IKKE1
C
C     SECOND LOOP SUROUNDING KKV3 IS DONE, THIRD LOOP OVER KKV1 START .
C
            IVV = KKV1
            IE = IABS( IICOLR( IJTRIG + 1 ) )
            IF( JE( 5 , IE ) . NE . 0 ) THEN
            IER = IE
            GO TO 570
            END IF
            IV1 = JE( 1 , IE )
            IF( IV1 . EQ . IVV ) THEN
            ISI = JE( 3 , IE )
            ELSE
            ISI = JE( 4 , IE )
            END IF
            IS = ISI
            ISI = 0
C
580         CONTINUE
C
            ITRIG = ITRIG + 1
            ISCRSS( ITRIG ) = IS
            IETRIG = IETRIG + 1
            IECRSS( IETRIG ) = IE
C
            IF(
     .          HYDFLX( IS , 4 ) . GT . FLUXPP . OR .
     .          HYDFLX( IS , 2 ) . GT . FLUXUU . OR .
     .          HYDFLX( IS , 1 ) . GT . FLUXRR . OR .
     .          KSDELT( IS ) . GT . NIDUMP . OR .
     .          XS( 3 , IS ) . GT . AREVGG ) THEN
            INDCTR = 3
            RETURN
            END IF
C
            DO 590 IR = 1 , 3
            JR = MOD( IR , 3 ) + 1
            IEA = IABS( JS( JR + 3 , IS ) )
            IF( IEA . EQ . IE ) THEN
            IIR = MOD( JR , 3 ) + 4
            IEI = JS( IIR , IS )
```

```
                IEIB = IABS( IEI )
                XEIEB = XE( 1 , IEIB )
                XYLNGT = XYLNGT + XEIEB
                IF( XYLONG . LT . XEIEB ) XYLONG = XEIEB
                IF( XYSHRT . GT . XEIEB ) XYSHRT = XEIEB
                IJTRIG = IJTRIG + 1
                IICOLR( IJTRIG ) = IEI
                JJR = MOD( JR + 1 , 3 ) + 4
                IER = IABS( JS( JJR , IS ) )
C
                IV1 = JE( 1 , IER )
                IF( IV1 . EQ . IVV ) THEN
                ISR = JE( 3 , IER )
                ELSE
                ISR = JE( 4 , IER )
                END IF
                END IF
C
590             CONTINUE
C
                IF( ISR . NE . ISI ) THEN
                IS = ISR
                IE = IER
                GO TO 580
                END IF
C
570             CONTINUE
C
                IETRIG = IETRIG + 1
                IECRSS( IETRIG ) = IER
C
                ITYPE = JE( 5 , IER )
C
                XEIEB = XE( 1 , IER )
                XEIEB = XXYYIB + XEIEB
                XYLNGT = XYLNGT + XEIEB
                IF( XYLONG . LT . XEIEB ) XYLONG = XEIEB
                IF( XYSHRT . GT . XEIEB ) XYSHRT = XEIEB
C
                INDCTR = 2
C                IF( XYLONG / XYSHRT . GT . 10. . AND . JLOOP . EQ . 0 ) RETURN
C
                JE( 2 , IEJKK ) = IVIN2
C
                IV1 = IVIN1
                IE1 = IICOLR( IJTRIG )
                IF( IE1 . GT . 0 ) THEN
                IV2 = JE( 2 , IE1 )
                ELSE
                IV2 = JE( 1 , - IE1 )
                END IF
C
                NEC = IECRSS( IETRIG )
                IETRIG = IETRIG - 1
C
                JV( 2 , IV2 ) = - NEC
                JE( 1 , NEC ) = IV2
                JE( 2 , NEC ) = IV1
                JE( 4 , NEC ) = 0
                JE( 5 , NEC ) = ITYPE
C
                IJTRIG = IJTRIG + 1
                IICOLR( IJTRIG ) = NEC
C
                ELSE IF( IKKE . EQ . 5 ) THEN
                  print*,'ikke=5',ksd,ikke
C
C     BEGINING THE DELETION PROCESS IF KSD HAS AN EDGE ON THE BOUNDARY
C     AND THE THIRD VERTEX IS OLSO ON THE BOUNDARY.
C     THE FIRST LOOP IS AROUND VERTEX KKV3 .
C
                IVV = KKV3
                IE = IEIN2
                XXYYIB = XE( 1 , IE )
```

```fortran
      IV1 = JE( 1 , IE )
      IVIN1 = JE( 2 , IE )
      IF( IV1 . EQ . IVV ) THEN
      ISI = JE( 3 , IE )
      ELSE
      ISI = JE( 4 , IE )
      END IF
      IS = ISI
C
600   CONTINUE
C
      ITRIG = ITRIG + 1
      ISCRSS( ITRIG ) = IS
      IETRIG = IETRIG + 1
      IECRSS( IETRIG ) = IE
C
      IF(
     .     HYDFLX( IS , 4 ) . GT . FLUXPP . OR .
     .     HYDFLX( IS , 2 ) . GT . FLUXUU . OR .
     .     HYDFLX( IS , 1 ) . GT . FLUXRR . OR .
     .     KSDELT( IS ) . GT . NIDUMP . OR .
     .     XS( 3 , IS ) . GT . AREVGG ) THEN
      INDCTR = 3
      RETURN
      END IF
C
      DO 610 IR = 1 , 3
      JR = MOD( IR , 3 ) + 1
      IEA = IABS( JS( JR + 3 , IS ) )
      IF( IEA . EQ . IE ) THEN
      IIR = MOD( JR , 3 ) + 4
      IEI = JS( IIR , IS )
      IEIB = IABS( IEI )
      XEIEB = XE( 1 , IEIB )
      XYLNGT = XYLNGT + XEIEB
      IF( XYLONG . LT . XEIEB ) XYLONG = XEIEB
      IF( XYSHRT . GT . XEIEB ) XYSHRT = XEIEB
      IJTRIG = IJTRIG + 1
      IICOLR( IJTRIG ) = IEI
      JJR = MOD( JR + 1 , 3 ) + 4
      IER = IABS( JS( JJR , IS ) )
C
      IV1 = JE( 1 , IER )
      IF( IV1 . EQ . IVV ) THEN
      ISR = JE( 3 , IER )
      ELSE
      ISR = JE( 4 , IER )
      END IF
      END IF
C
610   CONTINUE
C
      IF( IER . NE . IKKE2 ) THEN
      IS = ISR
      IE = IER
      GO TO 600
      END IF
C
C    FIRST LOOP SUROUNDING KKV3 IS DONE, SECOND LOOP OVER KKV2 START .
C
      IJTRIG = IJTRIG - 1
620   CONTINUE
C
      IEJK = IICOLR( IJTRIG )
      IF( IEJK . GT . 0 ) THEN
      IVIEJK = JE( 1 , IEJK )
      IJEJK5 = JE( 5 , IEJK )
      ELSE
      IVIEJK = JE( 2 , -IEJK )
      IJEJK5 = JE( 5 , -IEJK )
      END IF
C
      IF( IJEJK5 . EQ . 0 ) THEN
C
```

```
C    INTERMEDIATE LOOP START .
C
          JLOOP = 1
          IEJKI = IABS( IICOLR( IJTRIG - 1 ) )
          IEJK2 = IABS( IEJK )
          IETRIG = IETRIG + 1
          IECRSS( IETRIG ) = IEJK2
          IJTRIG = IJTRIG - 2
          IVV = IVIEJK
          JVDELT = JVDELT + 1
          IVDELT( JVDELT ) = IVV
          IE = IEJKI
          IV1 = JE( 1 , IE )
          IF( IV1 . EQ . IVV ) THEN
          ISI = JE( 3 , IE )
          ELSE
          ISI = JE( 4 , IE )
          END IF
          IS = ISI
          IET = IEJK2
C
630       CONTINUE
C
          ITRIG = ITRIG + 1
          ISCRSS( ITRIG ) = IS
C
          IETRIG = IETRIG + 1
          IECRSS( IETRIG ) = IE
C
          IF(
     .       HYDFLX( IS , 4 ) . GT . FLUXPP . OR .
     .       HYDFLX( IS , 2 ) . GT . FLUXUU . OR .
     .       HYDFLX( IS , 1 ) . GT . FLUXRR . OR .
     .       KSDELT( IS ) . GT . NIDUMP . OR .
     .       XS( 3 , IS ) . GT . AREVGG ) THEN
          INDCTR = 3
          RETURN
          END IF
C
          DO 640 IR = 1 , 3
          JR = MOD( IR , 3 ) + 1
          IEA = IABS( JS( JR + 3 , IS ) )
          IF( IEA . EQ . IE ) THEN
          IIR = MOD( JR , 3 ) + 4
          IEI = JS( IIR , IS )
          IEIB = IABS( IEI )
          XEIEB = XE( 1 , IEIB )
          XYLNGT = XYLNGT + XEIEB
          IF( XYLONG . LT . XEIEB ) XYLONG = XEIEB
          IF( XYSHRT . GT . XEIEB ) XYSHRT = XEIEB
          IIKK = IABS( IICOLR( IJTRIG ) )
          IF( IIKK . EQ . IEIB ) THEN
          JLOOP = 2
          IETRIG = IETRIG + 1
          IECRSS( IETRIG ) = IEIB
          IJTRIG = IJTRIG - 1
          IF( IEI . GT . 0 ) THEN
          JKVV = JE( 1 , IEIB )
          ELSE
          JKVV = JE( 2 , IEIB )
          END IF
          JVDELT = JVDELT + 1
          IVDELT( JVDELT ) = JKVV
          ELSE
          IJTRIG = IJTRIG + 1
          IICOLR( IJTRIG ) = IEI
          END IF
          JJR = MOD( JR + 1 , 3 ) + 4
          IER = IABS( JS( JJR , IS ) )
C
          IV1 = JE( 1 , IER )
          IF( IV1 . EQ . IVV ) THEN
          ISR = JE( 3 , IER )
          ELSE
```

```
                ISR = JE( 4 , IER )
                END IF
                END IF
C
640             CONTINUE
C
                IF( IER . NE . IET ) THEN
                IS = ISR
                IE = IER
                GO TO 630
                END IF
C
                GO TO 620
                END IF
C
C       INTERMEDIATE LOOP IS DONE, SECOND LOOP OVER KKV2 START .
C
                IVV = KKV2
                IE = IEIN1
                XXYYIC = XE( 1 , IE ) + XE( 1 , IKKE1 ) + XE( 1 , IEIB )
                XYLNGT = XYLNGT + XXYYIC - XE( 1 , IEIB )
                IF( XYLONG . LT . XXYYIC ) XYLONG = XXYYIC
                IF( XYSHRT . GT . XXYYIC ) XYSHRT = XXYYIC
                IVIN2 = JE( 2 , IE )
                IEJKK = IICOLR( IJTRIG )
                IV1 = JE( 1 , IE )
                IF( IV1 . EQ . IVV ) THEN
                ISI = JE( 3 , IE )
                ELSE
                ISI = JE( 4 , IE )
                END IF
                IS = ISI
C
650             CONTINUE
C
                ITRIG = ITRIG + 1
                ISCRSS( ITRIG ) = IS
C
                IETRIG = IETRIG + 1
                IECRSS( IETRIG ) = IE
C
                IF(
            .        HYDFLX( IS , 4 ) . GT . FLUXPP . OR .
            .        HYDFLX( IS , 2 ) . GT . FLUXUU . OR .
            .        HYDFLX( IS , 1 ) . GT . FLUXRR . OR .
            .        KSDELT( IS ) . GT . NIDUMP . OR .
            .        XS( 3 , IS ) . GT . AREVGG ) THEN
                INDCTR = 3
                RETURN
                END IF
C
                DO 660 IR = 1 , 3
                JR = MOD( IR , 3 ) + 1
                IEA = IABS( JS( JR + 3 , IS ) )
                IF( IEA . EQ . IE ) THEN
                IIR = MOD( JR , 3 ) + 4
                IEI = JS( IIR , IS )
                IEIB = IABS( IEI )
                XEIEB = XE( 1 , IEIB )
                XYLNGT = XYLNGT + XEIEB
                IF( XYLONG . LT . XEIEB ) XYLONG = XEIEB
                IF( XYSHRT . GT . XEIEB ) XYSHRT = XEIEB
                IJTRIG = IJTRIG + 1
                IICOLR( IJTRIG ) = IEI
                JJR = MOD( JR + 1 , 3 ) + 4
                IER = IABS( JS( JJR , IS ) )
C
                IV1 = JE( 1 , IER )
                IF( IV1 . EQ . IVV ) THEN
                ISR = JE( 3 , IER )
                ELSE
                ISR = JE( 4 , IER )
                END IF
                END IF
```

```
C
660       CONTINUE
C
          IF( IER . NE . IKKE2 ) THEN
          IS = ISR
          IE = IER
          GO TO 650
          END IF
C
          IJTRIG = IJTRIG - 1
          IETRIG = IETRIG + 1
          IECRSS( IETRIG ) = IKKE2
          IETRIG = IETRIG + 1
          IECRSS( IETRIG ) = IKKE1
C
C    SECOND LOOP SUROUNDING KKV2 IS DONE, THIRD LOOP OVER KKV3 START .
C
          IVV = KKV3
          IE = IABS( IICOLR( IJTRIG + 1 ) )
          IF( JE( 5 , IE ) . NE . 0 ) THEN
          IER = IE
          GO TO 670
          END IF
          IV1 = JE( 1 , IE )
          IF( IV1 . EQ . IVV ) THEN
          ISI = JE( 3 , IE )
          ELSE
          ISI = JE( 4 , IE )
          END IF
          IS = ISI
          ISI = 0
C
680       CONTINUE
C
          ITRIG = ITRIG + 1
          ISCRSS( ITRIG ) = IS
          IETRIG = IETRIG + 1
          IECRSS( IETRIG ) = IE
C
          IF(
     .        HYDFLX( IS , 4 ) . GT . FLUXPP . OR .
     .        HYDFLX( IS , 2 ) . GT . FLUXUU . OR .
     .        HYDFLX( IS , 1 ) . GT . FLUXRR . OR .
     .        KSDELT( IS ) . GT . NIDUMP . OR .
     .        XS( 3 , IS ) . GT . AREVGG ) THEN
          INDCTR = 3
          RETURN
          END IF
C
          DO 690 IR = 1 , 3
          JR = MOD( IR , 3 ) + 1
          IEA = IABS( JS( JR + 3 , IS ) )
          IF( IEA . EQ . IE ) THEN
          IIR = MOD( JR , 3 ) + 4
          IEI = JS( IIR , IS )
          IEIB = IABS( IEI )
          XEIEB = XE( 1 , IEIB )
          XYLNGT = XYLNGT + XEIEB
          IF( XYLONG . LT . XEIEB ) XYLONG = XEIEB
          IF( XYSHRT . GT . XEIEB ) XYSHRT = XEIEB
          IJTRIG = IJTRIG + 1
          IICOLR( IJTRIG ) = IEI
          JJR = MOD( JR + 1 , 3 ) + 4
          IER = IABS( JS( JJR , IS ) )
C
          IV1 = JE( 1 , IER )
          IF( IV1 . EQ . IVV ) THEN
          ISR = JE( 3 , IER )
          ELSE
          ISR = JE( 4 , IER )
          END IF
          END IF
C
690       CONTINUE
```

```
C
            IF( ISR . NE . ISI ) THEN
            IS = ISR
            IE = IER
            GO TO 680
            END IF
C
670         CONTINUE
C
            IETRIG = IETRIG + 1
            IECRSS( IETRIG ) = IER
C
            ITYPE = JE( 5 , IER )
C
            XEIEB = XE( 1 , IER )
            XEIEB = XXYYIB + XEIEB
            XYLNGT = XYLNGT + XEIEB
            IF( XYLONG . LT . XEIEB ) XYLONG = XEIEB
            IF( XYSHRT . GT . XEIEB ) XYSHRT = XEIEB
C
            INDCTR = 2
C            IF( XYLONG / XYSHRT . GT . 10. . AND . JLOOP . EQ . 0 ) RETURN
C
            JE( 2 , IEJKK ) = IVIN2
C
            IV1 = IVIN1
            IE1 = IICOLR( IJTRIG )
            IF( IE1 . GT . 0 ) THEN
            IV2 = JE( 2 , IE1 )
            ELSE
            IV2 = JE( 1 , - IE1 )
            END IF
C
            NEC = IECRSS( IETRIG )
            IETRIG = IETRIG - 1
C
            JV( 2 , IV2 ) = - NEC
            JE( 1 , NEC ) = IV2
            JE( 2 , NEC ) = IV1
            JE( 4 , NEC ) = 0
            JE( 5 , NEC ) = ITYPE
C
            IJTRIG = IJTRIG + 1
            IICOLR( IJTRIG ) = NEC
C
            END IF
C
C     LOOP OVER TRIANGLE KSD IS DONE
C
C     ELIMINATING THE DELETED TRIANGLES FROM JSDELT ARRAY
C
            LSDELT = 0
            DO 1520 IS = 1 , ISDELT
            JSP = JSDELT( IS )
            ILOOP = 0
            IF( JSP . EQ . 0 ) THEN
            ILOOP = 1
            ELSE
            DO 1525 IKS = 1 , ITRIG
            ISP = ISCRSS( IKS )
            IF( JSP . EQ . ISP ) ILOOP = 1
1525        CONTINUE
            END IF
            IF( ILOOP . EQ . 0 ) THEN
            LSDELT = LSDELT + 1
            JSDELT( LSDELT ) = JSP
            END IF
1520        CONTINUE
            ISDELT = LSDELT
C
            JVDELT = JVDELT + 1
            IVDELT( JVDELT ) = KV1
            JVDELT = JVDELT + 1
            IVDELT( JVDELT ) = KV2
```

```
          JVDELT = JVDELT + 1
          IVDELT( JVDELT ) = KV3
C
          DO 700 IE = 1 , IJTRIG
          IEM = IABS( IICOLR( IE ) )
          JUE( IE ) = IEM
          IV1 = JE( 1 , IEM )
          IV2 = JE( 2 , IEM )
          IEE1 = JV( 2 , IV1 )
          IEE2 = JV( 2 , IV2 )
          IF( IEE1 . GT . 0 ) JV( 2 , IV1 ) = IEM
          IF( IEE2 . GT . 0 ) JV( 2 , IV2 ) = IEM
  700     CONTINUE
C
          JTRIG = IJTRIG
          ISTOP = 0
          NSINTL = 0
C
          JJTRIG = IJTRIG
          DO 710 IE = 1 , JTRIG
          IEM = IICOLR( IE )
          IF( IEM . GT . 0 ) THEN
          JUV( IE ) = JE( 1 , IEM )
          ELSE
          JUV( IE ) = JE( 2 , - IEM )
          END IF
          IITRIG( IE ) = JUV( IE )
  710     CONTINUE
C
  720     CONTINUE
C
          JTRIGP = JTRIG + 1
          DO 730 IE = 1 , JTRIG
          IEM = IICOLR( IE )
          IF( IEM . GT . 0 ) THEN
          JUV( IE ) = JE( 1 , IEM )
          ELSE
          JUV( IE ) = JE( 2 , - IEM )
          END IF
  730     CONTINUE
C
          AREMIN = 1000000.
          IEMIN = 1
          DO 740 IE = 1 , JTRIG
C
          IEM = MOD( IE - 1 , JTRIG ) + 1
          IEP = MOD( IE , JTRIG ) + 1
          IEI = MOD( IE + 1 , JTRIG ) + 1
C
          IV1 = JUV( IEM )
          IV2 = JUV( IEP )
          IV3 = JUV( IEI )
C
          X1 = XV( 1 , IV1 ) - XV( 1 , IV2 )
          Y1 = XV( 2 , IV1 ) - XV( 2 , IV2 )
          X2 = XV( 1 , IV3 ) - XV( 1 , IV2 )
          Y2 = XV( 2 , IV3 ) - XV( 2 , IV2 )
          XSIN = ( X2 * Y1 - X1 * Y2 )
          XCOS = ( X1 * X2 + Y1 * Y2 )
          XCOT = XCOS / ( XSIN + 1.E-8 )
          IF( XSIN . LT . 0. . AND . XCOT . LT . AREMIN ) THEN
          AREMIN = XCOT
          IEMIN = IE
          END IF
          ANGLE( IE ) = XSIN / ( ABS( XCOS ) + 1.E-7 )
C
  740     CONTINUE
C
          DO 750 IE = 1 , JTRIG
          IEP = MOD( IE - IEMIN + JTRIG , JTRIG ) + 1
          JEN( IEP ) = IICOLR( IE )
          ANGLER( IEP ) = ANGLE( IE )
  750     CONTINUE
C
```

```
      DO 760 IE = 1 , JTRIG
      IICOLR( IE ) = JEN( IE )
      ANGLE( IE ) = ANGLER( IE )
  760 CONTINUE
C
      IFINAL = 0
      IEI = 1
      DO 770 IE = 1 , JTRIG
      SANGLE = ANGLE( IE )
      IANGLE( IE ) = - 1
      IF( SANGLE . GT . 1.E-2 ) IANGLE( IE ) = 1
  770 CONTINUE
C
      DO 780 IE = 1 , JTRIG
      IEM = MOD( IE - 1 , JTRIG ) + 1
      IEP = MOD( IE , JTRIG ) + 1
      IKM = MOD( IE + 1 , JTRIG ) + 1
      KEM = IANGLE( IEM )
      KEP = IANGLE( IEP )
      KKM = IANGLE( IKM )
      IF( KEM . EQ . - 1 . AND .
     .    KEP . EQ . 1 . AND .
     .    KKM . EQ . - 1 . AND . IFINAL . EQ . 0 ) THEN
      IEI = IKM
      IFINAL = 1
      END IF
  780 CONTINUE
C
      IF( IFINAL . EQ .  0 ) THEN
      DO 790 IE = 1 , JTRIG
      IEM = MOD( IE - 1 , JTRIG ) + 1
      IEP = MOD( IE , JTRIG ) + 1
      KEM = IANGLE( IEM )
      KEP = IANGLE( IEP )
      IF( KEM . EQ . - 1 . AND . KEP . EQ . 1 . AND .
     .                           IFINAL . EQ . 0 ) THEN
      IEI = MOD( IE + 1 , JTRIG ) + 1
      IFINAL = 1
      END IF
  790 CONTINUE
      END IF
C
      IF( IFINAL . EQ .  0 ) THEN
      ANGMIN = 10000000.
      DO 800 IE = 1 , JTRIG
      XANGLE = ANGLE( IE )
      SANGLE = ABS( XANGLE - 1. )
      IF( XANGLE . GT . 0. . AND . SANGLE . LT . ANGMIN ) THEN
      IEI = MOD( IE , JTRIG ) + 1
      ANGMIN = SANGLE
      END IF
  800 CONTINUE
      END IF
C
      DO 810 IE = 1 , JTRIG
      IEP = MOD( IE - IEI + JTRIGP , JTRIG ) + 1
      JEN( IEP ) = IICOLR( IE )
      ANGLER( IEP ) = ANGLE( IE )
  810 CONTINUE
C
      DO 820 IE = 1 , JTRIG
      ANGLE( IE ) = ANGLER( IE )
      IICOLR( IE ) = JEN( IE )
  820 CONTINUE
C
      DO 830 IE = 1 , JTRIG
      IEM = JEN( IE )
      IF( IEM . GT . 0 ) THEN
      JUV( IE ) = JE( 1 , IEM )
      ELSE
      JUV( IE ) = JE( 2 , - IEM )
      END IF
  830 CONTINUE
C
```

```
          IF( JTRIG . EQ . 3 ) THEN
C
          NSC = ISCRSS( ITRIG )
          ITRIG = ITRIG - 1
          NSINTL = NSINTL + 1
          INVTRG( NSINTL ) = NSC
          JS( 1 , NSC ) = JUV( 1 )
          JS( 2 , NSC ) = JUV( 2 )
          JS( 3 , NSC ) = JUV( 3 )
          JS( 4 , NSC ) = JEN( 1 )
          JS( 5 , NSC ) = JEN( 2 )
          JS( 6 , NSC ) = JEN( 3 )
C
          IV1 = JS( 1 , NSC )
          IV2 = JS( 2 , NSC )
          IV3 = JS( 3 , NSC )
          AX = XV( 1 , IV2 ) - XV( 1 , IV1 )
          AY = XV( 2 , IV2 ) - XV( 2 , IV1 )
          BX = XV( 1 , IV3 ) - XV( 1 , IV1 )
          BY = XV( 2 , IV3 ) - XV( 2 , IV1 )
          XS( 3 , NSC ) = 0.5 * ( AX * BY - AY * BX )
C
          SAREA( NSC ) = 1. / XS( 3 , NSC )
          XXC = ( XV( 1 , IV1 ) + XV( 1 , IV2 ) + XV( 1 , IV3 ) ) *
     .            THIRD
          YYC = ( XV( 2 , IV1 ) + XV( 2 , IV2 ) + XV( 2 , IV3 ) ) *
     .            THIRD
          XS( 1 , NSC ) = XXC
          XS( 2 , NSC ) = YYC
          HYDFLX( NSC , 4 ) = 0.
          HYDFLX( NSC , 1 ) = 0.
          HYDFLX( NSC , 2 ) = 0.
          KSDELT( NSC ) = 1
C
          DO 840 IR = 1 , MHQ
          HYDV( NSC , IR ) = ( HYDVVV( IV1 , IR ) +
     .                         HYDVVV( IV2 , IR ) +
     .                         HYDVVV( IV3 , IR ) ) * THIRD
  840     CONTINUE
C
          HDUM           = 1. / ( HYDV( NSC , 1 ) + 1.E-12 )
          HYDV( NSC , 2 ) = HYDV( NSC , 2 ) * HDUM
          HYDV( NSC , 3 ) = HYDV( NSC , 3 ) * HDUM
          HYDV( NSC , 4 ) = ( HYDV( NSC , 4 )  -
     .                   .5 * HYDV( NSC , 1 ) *
     .    ( HYDV( NSC , 2 ) * HYDV( NSC , 2 ) +
     .      HYDV( NSC , 3 ) * HYDV( NSC , 3 ) ) ) *
     .                     ( HYDV( NSC , 5 ) - 1. )
C
          ISTOP = 1
C
          ELSE IF( JTRIG . EQ . 4 ) THEN
C
          NSC = ISCRSS( ITRIG )
          ITRIG = ITRIG - 1
          NSINTL = NSINTL + 1
          INVTRG( NSINTL ) = NSC
          NEC = IECRSS( IETRIG )
          IETRIG = IETRIG - 1
C
          IJTRIG = IJTRIG + 1
          JUE( IJTRIG ) = NEC
C
          JE( 1 , NEC ) = JUV( 1 )
          JE( 2 , NEC ) = JUV( 3 )
          JE( 5 , NEC ) = 0
C
          JS( 1 , NSC ) = JUV( 1 )
          JS( 2 , NSC ) = JUV( 2 )
          JS( 3 , NSC ) = JUV( 3 )
          JS( 4 , NSC ) = JEN( 1 )
          JS( 5 , NSC ) = JEN( 2 )
          JS( 6 , NSC ) = - NEC
C
```

```
            NSC = ISCRSS( ITRIG )
            ITRIG = ITRIG - 1
            NSINTL = NSINTL + 1
            INVTRG( NSINTL ) = NSC
C
            JS( 1 , NSC ) = JUV( 1 )
            JS( 2 , NSC ) = JUV( 3 )
            JS( 3 , NSC ) = JUV( 4 )
            JS( 4 , NSC ) = NEC
            JS( 5 , NSC ) = JEN( 3 )
            JS( 6 , NSC ) = JEN( 4 )
C
            DO 850 IKR = 1 , 2
            NSS = INVTRG( NSINTL + 1 - IKR )
            IV1 = JS( 1 , NSS )
            IV2 = JS( 2 , NSS )
            IV3 = JS( 3 , NSS )
            AX = XV( 1 , IV2 ) - XV( 1 , IV1 )
            AY = XV( 2 , IV2 ) - XV( 2 , IV1 )
            BX = XV( 1 , IV3 ) - XV( 1 , IV1 )
            BY = XV( 2 , IV3 ) - XV( 2 , IV1 )
            XS( 3 , NSS ) = 0.5 * ( AX * BY - AY * BX )
C
            SAREA( NSS ) = 1. / XS( 3 , NSS )
            XXC = ( XV( 1 , IV1 ) + XV( 1 , IV2 ) + XV( 1 , IV3 ) ) *
     .              THIRD
            YYC = ( XV( 2 , IV1 ) + XV( 2 , IV2 ) + XV( 2 , IV3 ) ) *
     .              THIRD
            XS( 1 , NSS ) = XXC
            XS( 2 , NSS ) = YYC
            HYDFLX( NSS , 4 ) = 0.
            HYDFLX( NSS , 1 ) = 0.
            HYDFLX( NSS , 2 ) = 0.
            KSDELT( NSS ) = 1
C
            DO 860 IR = 1 , MHQ
            HYDV( NSS , IR ) = ( HYDVVV( IV1 , IR ) +
     .                           HYDVVV( IV2 , IR ) +
     .                           HYDVVV( IV3 , IR ) ) * THIRD
860        CONTINUE
C
            HDUM          = 1. / ( HYDV( NSS , 1 ) + 1.E-12 )
            HYDV( NSS , 2 ) = HYDV( NSS , 2 ) * HDUM
            HYDV( NSS , 3 ) = HYDV( NSS , 3 ) * HDUM
            HYDV( NSS , 4 ) = ( HYDV( NSS , 4 )  -
     .                     .5 * HYDV( NSS , 1 ) *
     .      ( HYDV( NSS , 2 ) * HYDV( NSS , 2 ) +
     .        HYDV( NSS , 3 ) * HYDV( NSS , 3 ) ) ) *
     .                       ( HYDV( NSS , 5 ) - 1. )
C
850         CONTINUE
            ISTOP = 1
C
            ELSE
C
            NSC = ISCRSS( ITRIG )
            ITRIG = ITRIG - 1
            NSINTL = NSINTL + 1
            INVTRG( NSINTL ) = NSC
            NEC = IECRSS( IETRIG )
            IETRIG = IETRIG - 1
C
            IJTRIG = IJTRIG + 1
            JUE( IJTRIG ) = NEC
C
            JE( 1 , NEC ) = JUV( 1 )
            JE( 2 , NEC ) = JUV( 3 )
            JE( 5 , NEC ) = 0
C
            JS( 1 , NSC ) = JUV( 1 )
            JS( 2 , NSC ) = JUV( 2 )
            JS( 3 , NSC ) = JUV( 3 )
            JS( 4 , NSC ) = JEN( 1 )
            JS( 5 , NSC ) = JEN( 2 )
```

```
            JS( 6 , NSC ) = - NEC
C
            IICOLR( 1 ) = NEC
            JTRIG = JTRIG - 1
            DO 870 IEE = 2 , JTRIG
            IICOLR( IEE ) = JEN( IEE + 1 )
870         CONTINUE
C
            IV1 = JS( 1 , NSC )
            IV2 = JS( 2 , NSC )
            IV3 = JS( 3 , NSC )
            AX = XV( 1 , IV2 ) - XV( 1 , IV1 )
            AY = XV( 2 , IV2 ) - XV( 2 , IV1 )
            BX = XV( 1 , IV3 ) - XV( 1 , IV1 )
            BY = XV( 2 , IV3 ) - XV( 2 , IV1 )
            XS( 3 , NSC ) = 0.5 * ( AX * BY - AY * BX )
C
            SAREA( NSC ) = 1. / XS( 3 , NSC )
            XXC = ( XV( 1 , IV1 ) + XV( 1 , IV2 ) + XV( 1 , IV3 ) ) *
     .            THIRD
            YYC = ( XV( 2 , IV1 ) + XV( 2 , IV2 ) + XV( 2 , IV3 ) ) *
     .            THIRD
            XS( 1 , NSC ) = XXC
            XS( 2 , NSC ) = YYC
            HYDFLX( NSC , 4 ) = 0.
            HYDFLX( NSC , 1 ) = 0.
            HYDFLX( NSC , 2 ) = 0.
            KSDELT( NSC ) = 1
C
            DO 880 IR = 1 , MHQ
            HYDV( NSC , IR ) = ( HYDVVV( IV1 , IR ) +
     .                           HYDVVV( IV2 , IR ) +
     .                           HYDVVV( IV3 , IR ) ) * THIRD
880         CONTINUE
C
            HDUM           = 1. / ( HYDV( NSC , 1 ) + 1.E-12 )
            HYDV( NSC , 2 ) = HYDV( NSC , 2 ) * HDUM
            HYDV( NSC , 3 ) = HYDV( NSC , 3 ) * HDUM
            HYDV( NSC , 4 ) = ( HYDV( NSC , 4 )  -
     .                     .5 * HYDV( NSC , 1 ) *
     .      ( HYDV( NSC , 2 ) * HYDV( NSC , 2 ) +
     .        HYDV( NSC , 3 ) * HYDV( NSC , 3 ) ) ) *
     .                         ( HYDV( NSC , 5 ) - 1. )
C
            END IF
            IF( ISTOP . EQ . 0 ) GO TO 720
C
            DO 890 ISS = 1 , NSINTL
            IS = INVTRG( ISS )
            DO 890 IR = 4 , 6
            IE = JS( IR , IS )
            IF( IE . GT . 0 ) THEN
            JE( 3 , IE ) = IS
            ELSE
            JE( 4 , - IE ) = IS
            END IF
890         CONTINUE
C
            DO 900 IENN = 1 , IJTRIG
            IEN = JUE( IENN )
            JV1 = JE( 1 , IEN )
            JV2 = JE( 2 , IEN )
            AX = XV( 1 , JV2 ) - XV( 1 , JV1 )
            AY = XV( 2 , JV2 ) - XV( 2 , JV1 )
            XE( 1 , IEN ) = SQRT( AX * AX + AY * AY )
            XEREV = 1. / XE( 1 , IEN )
            XN( IEN ) = AY * XEREV
            YN( IEN ) = - AX * XEREV
            ISSR = JE( 4 , IEN )
            ISSL = JE( 3 , IEN )
C
            IF( JE( 5 , IEN ) . NE . 0 ) THEN
C
            AA = XV( 1 , JV2 ) - XV( 1 , JV1 )
```

```
            BB = XV( 2 , JV2 ) - XV( 2 , JV1 )
            XEL = XS( 1 , ISSL )
            YEL = XS( 2 , ISSL )
            CC = XEL - XV( 1 , JV1 )
            DD = YEL - XV( 2 , JV1 )
            EE = ( AA * CC + BB * DD ) * XEREV * XEREV
            XER = XV( 1 , JV1 ) + AA * EE
            YER = XV( 2 , JV1 ) + BB * EE
            AX = XER - XEL
            AY = YER - YEL
            XE( 2 , IEN ) = SQRT( AX * AX + AY * AY )
            XEREV = 1. / XE( 2 , IEN )
            XXN( IEN ) = AX * XEREV
            YYN( IEN ) = AY * XEREV
            XE( 2 , IEN ) = 2. * XE( 2 , IEN )
            XYMIDL( IEN ) = .5
            XMIDL( IEN ) = XER
            YMIDL( IEN ) = YER
C
            ELSE
C
            XER = XS( 1 , ISSR )
            YER = XS( 2 , ISSR )
            XEL = XS( 1 , ISSL )
            YEL = XS( 2 , ISSL )
C
            AA = XV( 1 , JV2 ) - XV( 1 , JV1 )
            BB = XV( 2 , JV2 ) - XV( 2 , JV1 )
            CC = XEL - XER
            DD = YEL - YER
            ACA = XER - XV( 1 , JV1 )
            DBD = YER - XV( 2 , JV1 )
            EE = ( ACA * DD - DBD * CC ) / ( AA * DD - BB * CC )
            XMIDL( IEN ) = XV( 1 , JV1 ) + AA * EE
            YMIDL( IEN ) = XV( 2 , JV1 ) + BB * EE
C
            XEMID = XMIDL( IEN ) - XEL
            YEMID = YMIDL( IEN ) - YEL
C
            AX = XER - XEL
            AY = YER - YEL
            XE( 2 , IEN ) = SQRT( AX * AX + AY * AY )
            XEREV = 1. / XE( 2 , IEN )
            XXN( IEN ) = AX * XEREV
            YYN( IEN ) = AY * XEREV
C
            XYMIDL( IEN ) = SQRT( XEMID * XEMID + YEMID * YEMID ) * XEREV
C
            END IF
C
900         CONTINUE
C
C    ORDER THE DELETED VERTECIS IN A DECENDED ORDER IN AN ARRAY
C    NVDELT
C
            KFLIP = JVDELT
            DO 910 KK = 1 ,  JVDELT
            IFLIP = 1
            NVDELT( KK ) = IVDELT( 1 )
            DO 920 KI = 1 , KFLIP
            IF( IVDELT( KI ) . GT . NVDELT( KK ) ) THEN
            NVDELT( KK ) = IVDELT( KI )
            IFLIP = KI
            END IF
920         CONTINUE
            ISS = 0
            DO 930 KI = 1 , KFLIP
            IF( KI . NE . IFLIP ) THEN
            ISS = ISS + 1
            IVDELT( ISS ) = IVDELT( KI )
            END IF
930         CONTINUE
            KFLIP = KFLIP - 1
910         CONTINUE
```

```
C
C     ORDER THE DELETED EDGES IN A DECENDED ORDER IN AN ARRAY
C     NECRSS
C
            KFLIP = IETRIG
            DO 940 KK = 1 , IETRIG
            IFLIP = 1
            NECRSS( KK ) = IECRSS( 1 )
            DO 950 KI = 1 , KFLIP
            IF( IECRSS( KI ) . GT . NECRSS( KK ) ) THEN
            NECRSS( KK ) = IECRSS( KI )
            IFLIP = KI
            END IF
  950       CONTINUE
            ISS = 0
            DO 960 KI = 1 , KFLIP
            IF( KI . NE . IFLIP ) THEN
            ISS = ISS + 1
            IECRSS( ISS ) = IECRSS( KI )
            END IF
  960       CONTINUE
            KFLIP = KFLIP - 1
  940       CONTINUE
C
C     ORDER THE DELETED CELLS IN A DECENDED ORDER IN AN ARRAY
C     NSCRSS
C
            KFLIP = ITRIG
            DO 970 KK = 1 , ITRIG
            IFLIP = 1
            NSCRSS( KK ) = ISCRSS( 1 )
            DO 980 KI = 1 , KFLIP
            IF( ISCRSS( KI ) . GT . NSCRSS( KK ) ) THEN
            NSCRSS( KK ) = ISCRSS( KI )
            IFLIP = KI
            END IF
  980       CONTINUE
            ISS = 0
            DO 990 KI = 1 , KFLIP
            IF( KI . NE . IFLIP ) THEN
            ISS = ISS + 1
            ISCRSS( ISS ) = ISCRSS( KI )
            END IF
  990       CONTINUE
            KFLIP = KFLIP - 1
  970       CONTINUE
C
            DO 1000 KI = 1 , JVDELT
            IVDELT( KI ) = NV + 1 - KI
 1000       CONTINUE
C
            DO 1010 KI = 1 , IETRIG
            IECRSS( KI ) = NE + 1 - KI
 1010       CONTINUE
C
            DO 1020 KI = 1 , ITRIG
            ISCRSS( KI ) = NS + 1 - KI
 1020       CONTINUE
C
C     IT MAKE SURE THAT VERTICES THAT ARE TO BE DELETED ARE NOT
C     REPLACED BY VERTICES THAS ARE TO BE DELETED ALSO
C
            DO 1030 KI = 1 , JVDELT
            IVM = NVDELT( KI )
            DO 1030 KK = 1 , JVDELT
            JVM = IVDELT( KK )
            IF( IVM . EQ . JVM . AND . KK . NE . KI ) THEN
            IVDUM = IVDELT( KI )
            IVDELT( KI ) = IVM
            IVDELT( KK ) = IVDUM
            END IF
 1030       CONTINUE
C
C     IT MAKE SURE THAT EDGES THAT ARE TO BE DELETED ARE NOT
```

```
C     REPLACED BY EDGES THAS ARE TO BE DELETED ALSO
C
            DO 1040 KI = 1 , IETRIG
            IEM = NECRSS( KI )
            DO 1040 KK = 1 , IETRIG
            JEM = IECRSS( KK )
            IF( IEM . EQ . JEM . AND . KK . NE . KI ) THEN
            IEDUM = IECRSS( KI )
            IECRSS( KI ) = IEM
            IECRSS( KK ) = IEDUM
            END IF
 1040       CONTINUE
C
C     IT MAKE SURE THAT CELLS THAT ARE TO BE DELETED ARE NOT
C     REPLACED BY CELLS THAS ARE TO BE DELETED ALSO
C
            DO 1050 KI = 1 , ITRIG
            ISM = NSCRSS( KI )
            DO 1050 KK = 1 , ITRIG
            JSM = ISCRSS( KK )
            IF( ISM . EQ . JSM . AND . KK . NE . KI ) THEN
            ISDUM = ISCRSS( KI )
            ISCRSS( KI ) = ISM
            ISCRSS( KK ) = ISDUM
            END IF
 1050       CONTINUE
C
C    IVDELT(*) SEQUENCE OF VERTICES TO BE DELETED END OF LIST
C    NVDELT(*) SEQUENCE OF VERTICES TO BE REPLACED CURRENT IN LIST
C    ISCRSS(*) SEQUENCE OF TRIANGLES TO BE DELETED END OF LIST
C    NSCRSS(*) SEQUENCE OF TRIANGLES TO BE REPLACED CURRENT IN LIST
C    IECRSS(*) SEQUENCE OF EDGES TO BE DELETED END OF LIST
C    NECRSS(*) SEQUENCE OF EDGES TO BE REPLACED CURRENT IN LIST
C
            DO 1060 KI = 1 , JVDELT
            IVM = NVDELT( KI )
            JVM = IVDELT( KI )
C
            XV( 1 , IVM ) = XV( 1 , JVM )
            XV( 2 , IVM ) = XV( 2 , JVM )
            JV( 1 , IVM ) = JV( 1 , JVM )
C
            DO 1060 IR = 1 , MHQ
            HYDVVV( IVM , IR ) = HYDVVV( JVM , IR )
 1060       CONTINUE
C
            NVM = NV - JVDELT
            NEM = NE - IETRIG
            NSM = NS - ITRIG
C
C     UPDATE THE EDGES AND CELLS THAT ARE CONNECTED TO THE DELETED
C     VERTICES
C
            JNVEDG = 0
            JNVTRG = 0
            DO 1070 JVDL = 1 , JVDELT
            IVDL = NVDELT( JVDL )
            NVDL = IVDELT( JVDL )
            IF( IVDL . NE . NVDL ) THEN
            IE = JV( 2 , NVDL )
            IF( IE . GT . 0 ) THEN
C
            IV1 = JE( 1 , IE )
            IF( IV1 . EQ . NVDL ) THEN
            ISI = JE( 3 , IE )
            ELSE
            ISI = JE( 4 , IE )
            END IF
            IS = ISI
C
            JNVEDG = JNVEDG + 1
            INVEDG( JNVEDG ) = IE
            JNVTRG = JNVTRG + 1
            INVTRG( JNVTRG ) = IS
```

```
C
 1090 CONTINUE
C
      DO 1080 IR = 1 , 3
      JR = MOD( IR , 3 ) + 1
      IEA = IABS( JS( JR + 3 , IS ) )
      IF( IEA . EQ . IE ) THEN
      JJR = MOD( JR + 1 , 3 ) + 4
      IER = IABS( JS( JJR , IS ) )
C
      IV1 = JE( 1 , IER )
      IF( IV1 . EQ . NVDL ) THEN
      ISR = JE( 3 , IER )
      ELSE
      ISR = JE( 4 , IER )
      END IF
      END IF
C
 1080 CONTINUE
C
      IF( ISR . NE . ISI ) THEN
      IS = ISR
      IE = IER
C
      JNVEDG = JNVEDG + 1
      INVEDG( JNVEDG ) = IE
      JNVTRG = JNVTRG + 1
      INVTRG( JNVTRG ) = IS
C
      GO TO 1090
      END IF
C
      ELSE
C
      IE = - IE
      IV1 = JE( 1 , IE )
      IF( IV1 . EQ . NVDL ) THEN
      ISI = JE( 3 , IE )
      ELSE
      ISI = JE( 4 , IE )
      END IF
      IS = ISI
      ISI = 0
C
      JNVEDG = JNVEDG + 1
      INVEDG( JNVEDG ) = IE
      JNVTRG = JNVTRG + 1
      INVTRG( JNVTRG ) = IS
C
 1100 CONTINUE
C
      DO 1110 IR = 1 , 3
      JR = MOD( IR , 3 ) + 1
      IEA = IABS( JS( JR + 3 , IS ) )
      IF( IEA . EQ . IE ) THEN
      JJR = MOD( JR + 1 , 3 ) + 4
      IER = IABS( JS( JJR , IS ) )
C
      IV1 = JE( 1 , IER )
      IF( IV1 . EQ . NVDL ) THEN
      ISR = JE( 3 , IER )
      ELSE
      ISR = JE( 4 , IER )
      END IF
      END IF
C
 1110 CONTINUE
C
      IF( ISR . NE . ISI ) THEN
      IS = ISR
      IE = IER
C
      JNVEDG = JNVEDG + 1
      INVEDG( JNVEDG ) = IE
```

```
            JNVTRG = JNVTRG + 1
            INVTRG( JNVTRG ) = IS
C
            GO TO 1100
            END IF
C
            JNVEDG = JNVEDG + 1
            INVEDG( JNVEDG ) = IER
C
            END IF
            END IF
 1070       CONTINUE
C
            NSMNPT = INVTRG( 1 )
C
            DO 1120 IE = 1 , JNVEDG
            IEE = INVEDG( IE )
            DO 1120 IIDG = IE + 1 , JNVEDG
            IF( INVEDG( IIDG ) . EQ . IEE ) THEN
            INVEDG( IIDG ) = 0
            END IF
 1120       CONTINUE
C
            IEDUM = 0
            DO 1130 IIDG = 1 , JNVEDG
            IF( INVEDG( IIDG ) . NE . 0 ) THEN
            IEDUM = IEDUM + 1
            INVEDG( IEDUM ) = INVEDG( IIDG )
            END IF
 1130       CONTINUE
            JNVEDG = IEDUM
C
            DO 1140 IS = 1 , JNVTRG
            ISS = INVTRG( IS )
            DO 1140 IITG = IS + 1 , JNVTRG
            IF( INVTRG( IITG ) . EQ . ISS ) THEN
            INVTRG( IITG ) = 0
            END IF
 1140       CONTINUE
C
            ISDUM = 0
            DO 1150 IITG = 1 , JNVTRG
            IF( INVTRG( IITG ) . NE . 0 ) THEN
            ISDUM = ISDUM + 1
            INVTRG( ISDUM ) = INVTRG( IITG )
            END IF
 1150       CONTINUE
            JNVTRG = ISDUM
C
C     UPDATE THE VERTECIS AND CELLS THAT ARE CONNECTED TO THE DELETED
C     EDGES
C
            DO 1160 IE = 1 , IETRIG
            IES = IECRSS( IE )
C
            IV = JE( 1 , IES )
            IER = JV( 2 , IV )
            IIN = ISIGN( 1 , IER )
            IEE = IABS( IER )
            IEM = IEE
            DO 1170 KK = 1 , IETRIG
            JEM = IECRSS( KK )
            IF( IEE . EQ . JEM ) IEM = NECRSS( KK )
 1170       CONTINUE
            JV( 2 , IV ) = IIN * IEM
C
            IV = JE( 2 , IES )
            IER = JV( 2 , IV )
            IIN = ISIGN( 1 , IER )
            IEE = IABS( IER )
            IEM = IEE
            DO 1180 KK = 1 , IETRIG
            JEM = IECRSS( KK )
            IF( IEE . EQ . JEM ) IEM = NECRSS( KK )
```

```
 1180     CONTINUE
          JV( 2 , IV ) = IIN * IEM
C
 1160     CONTINUE
C
          DO 1190 KK = 1 , JJTRIG
          IVV = IITRIG( KK )
          DO 1190 JVDL = 1 , JVDELT
          IVDL = NVDELT( JVDL )
          NVDL = IVDELT( JVDL )
          IF( IVV . EQ . NVDL ) IITRIG( KK ) = IVDL
 1190     CONTINUE
C
          DO 1200 JVDL = 1 , JVDELT
          IVDL = NVDELT( JVDL )
          NVDL = IVDELT( JVDL )
          JV( 2 , IVDL ) = JV( 2 , NVDL )
 1200     CONTINUE
C
          DO 1210 IS = 1 , JNVTRG
          ISS = INVTRG( IS )
C
          IV = JS( 1 , ISS )
          IVM = IV
          DO 1220 KI = 1 , JVDELT
          JVM = IVDELT( KI )
          IF( IV . EQ . JVM ) IVM = NVDELT( KI )
 1220     CONTINUE
          JS( 1 , ISS ) = IVM
C
          IV = JS( 2 , ISS )
          IVM = IV
          DO 1230 KI = 1 , JVDELT
          JVM = IVDELT( KI )
          IF( IV . EQ . JVM ) IVM = NVDELT( KI )
 1230     CONTINUE
          JS( 2 , ISS ) = IVM
C
          IV = JS( 3 , ISS )
          IVM = IV
          DO 1240 KI = 1 , JVDELT
          JVM = IVDELT( KI )
          IF( IV . EQ . JVM ) IVM = NVDELT( KI )
 1240     CONTINUE
          JS( 3 , ISS ) = IVM
C
 1210     CONTINUE
C
          DO 1250 IE = 1 , JNVEDG
          IEE = INVEDG( IE )
C
          IV = JE( 1 , IEE )
          IVM = IV
          DO 1260 KI = 1 , JVDELT
          JVM = IVDELT( KI )
          IF( IV . EQ . JVM ) IVM = NVDELT( KI )
 1260     CONTINUE
          JE( 1 , IEE ) = IVM
C
          IV = JE( 2 , IEE )
          IVM = IV
          DO 1270 KI = 1 , JVDELT
          JVM = IVDELT( KI )
          IF( IV . EQ . JVM ) IVM = NVDELT( KI )
 1270     CONTINUE
          JE( 2 , IEE ) = IVM
C
 1250     CONTINUE
C
C     UPDATE THE VERTECIS AND EDGES THAT ARE CONNECTED TO THE DELETED
C     CELSS
C
          DO 1280 IS = 1 , ITRIG
C
```

```
           ISE = ISCRSS( IS )
C
           IE = IABS( JS( 4 , ISE ) )
           ISS = JE( 3 , IE )
           ISM = ISS
           DO 1290 KI = 1 , ITRIG
           JSM = ISCRSS( KI )
           IF( ISS . EQ . JSM ) ISM = NSCRSS( KI )
 1290      CONTINUE
           JE( 3 , IE ) = ISM
C
           ISS = JE( 4 , IE )
           ISM = ISS
           DO 1300 KI = 1 , ITRIG
           JSM = ISCRSS( KI )
           IF( ISS . EQ . JSM ) ISM = NSCRSS( KI )
 1300      CONTINUE
           JE( 4 , IE ) = ISM
C
           IE = IABS( JS( 5 , ISE ) )
           ISS = JE( 3 , IE )
           ISM = ISS
           DO 1310 KI = 1 , ITRIG
           JSM = ISCRSS( KI )
           IF( ISS . EQ . JSM ) ISM = NSCRSS( KI )
 1310      CONTINUE
           JE( 3 , IE ) = ISM
C
           ISS = JE( 4 , IE )
           ISM = ISS
           DO 1320 KI = 1 , ITRIG
           JSM = ISCRSS( KI )
           IF( ISS . EQ . JSM ) ISM = NSCRSS( KI )
 1320      CONTINUE
           JE( 4 , IE ) = ISM
C
           IE = IABS( JS( 6 , ISE ) )
           ISS = JE( 3 , IE )
           ISM = ISS
           DO 1330 KI = 1 , ITRIG
           JSM = ISCRSS( KI )
           IF( ISS . EQ . JSM ) ISM = NSCRSS( KI )
 1330      CONTINUE
           JE( 3 , IE ) = ISM
C
           ISS = JE( 4 , IE )
           ISM = ISS
           DO 1340 KI = 1 , ITRIG
           JSM = ISCRSS( KI )
           IF( ISS . EQ . JSM ) ISM = NSCRSS( KI )
 1340      CONTINUE
           JE( 4 , IE ) = ISM
C
 1280      CONTINUE
C
           DO 1350 IE = 1 , IETRIG
           IES = IECRSS( IE )
C
           IS = JE( 3 , IES )
           ISS = IS
           DO 1360 KI = 1 , ITRIG
           ISM = NSCRSS( KI )
           IF( IS . EQ . ISM ) ISS = ISCRSS( KI )
 1360      CONTINUE
C
           IF( ISS . NE . 0 ) THEN
C
           IER = JS( 4 , ISS )
           IEE = IABS( IER )
           IEM = IEE
           DO 1370 KI = 1 , IETRIG
           JEM = IECRSS( KI )
           IF( IEE . EQ . JEM ) IEM = NECRSS( KI )
 1370      CONTINUE
```

```
      JS( 4 , ISS ) = ISIGN( 1 , IER ) * IEM
C
      IER = JS( 5 , ISS )
      IEE = IABS( IER )
      IEM = IEE
      DO 1380 KI = 1 , IETRIG
      JEM = IECRSS( KI )
      IF( IEE . EQ . JEM ) IEM = NECRSS( KI )
 1380 CONTINUE
      JS( 5 , ISS ) = ISIGN( 1 , IER ) * IEM
C
      IER = JS( 6 , ISS )
      IEE = IABS( IER )
      IEM = IEE
      DO 1390 KI = 1 , IETRIG
      JEM = IECRSS( KI )
      IF( IEE . EQ . JEM ) IEM = NECRSS( KI )
 1390 CONTINUE
      JS( 6 , ISS ) = ISIGN( 1 , IER ) * IEM
C
      END IF
C
      IS = JE( 4 , IES )
      ISS = IS
      DO 1400 KI = 1 , ITRIG
      ISM = NSCRSS( KI )
      IF( IS . EQ . ISM ) ISS = ISCRSS( KI )
 1400 CONTINUE
C
      IF( ISS . NE . 0 ) THEN
C
      IER = JS( 4 , ISS )
      IEE = IABS( IER )
      IEM = IEE
      DO 1410 KI = 1 , IETRIG
      JEM = IECRSS( KI )
      IF( IEE . EQ . JEM ) IEM = NECRSS( KI )
 1410 CONTINUE
      JS( 4 , ISS ) = ISIGN( 1 , IER ) * IEM
C
      IER = JS( 5 , ISS )
      IEE = IABS( IER )
      IEM = IEE
      DO 1420 KI = 1 , IETRIG
      JEM = IECRSS( KI )
      IF( IEE . EQ . JEM ) IEM = NECRSS( KI )
 1420 CONTINUE
      JS( 5 , ISS ) = ISIGN( 1 , IER ) * IEM
C
      IER = JS( 6 , ISS )
      IEE = IABS( IER )
      IEM = IEE
      DO 1430 KI = 1 , IETRIG
      JEM = IECRSS( KI )
      IF( IEE . EQ . JEM ) IEM = NECRSS( KI )
 1430 CONTINUE
      JS( 6 , ISS ) = ISIGN( 1 , IER ) * IEM
C
      END IF
C
 1350 CONTINUE
C
      DO 1440 IE = 1 , IETRIG
      IEM = NECRSS( IE )
      JEM = IECRSS( IE )
C
      DO 1450 IK = 1 , 5
      JE( IK , IEM ) = JE( IK , JEM )
 1450 CONTINUE
C
      XE( 1 , IEM ) = XE( 1 , JEM )
      XE( 2 , IEM ) = XE( 2 , JEM )
C
      XN( IEM ) = XN( JEM )
```

```
            YN( IEM ) = YN( JEM )
            XXN( IEM ) = XXN( JEM )
            YYN( IEM ) = YYN( JEM )
            XMIDL( IEM ) = XMIDL( JEM )
            YMIDL( IEM ) = YMIDL( JEM )
            XYMIDL( IEM ) = XYMIDL( JEM )
 1440       CONTINUE
C
            DO 1460 IS = 1 , ITRIG
            ISM = NSCRSS( IS )
            JSM = ISCRSS( IS )
C
            DO 1470 IK = 1 , 6
            JS( IK , ISM ) = JS( IK , JSM )
 1470       CONTINUE
C
            XS( 1 , ISM ) = XS( 1 , JSM )
            XS( 2 , ISM ) = XS( 2 , JSM )
            XS( 3 , ISM ) = XS( 3 , JSM )
C
            SAREA( ISM ) = SAREA( JSM )
            KSDELT( ISM ) = KSDELT( JSM )
C
            DO 1480 IK = 1 , MHQ
            HYDV( ISM , IK ) = HYDV( JSM , IK )
 1480       CONTINUE
C
            HYDFLX( ISM , 4 ) = HYDFLX( JSM , 4 )
            HYDFLX( ISM , 1 ) = HYDFLX( JSM , 1 )
            HYDFLX( ISM , 2 ) = HYDFLX( JSM , 2 )
C
 1460       CONTINUE
C
            NV = NVM
            NE = NEM
            NS = NSM
C
            DO 1490 IENN = 1 , IJTRIG
            IE = JUE( IENN )
            DO 1490 KI = 1 , IETRIG
            JEM = IECRSS( KI )
            IF( IE . EQ . JEM ) JUE( IENN ) = NECRSS( KI )
 1490       CONTINUE
C
            DO 1540 IENN = 1 , JJTRIG
            IVV = IITRIG( IENN )
            IF( JV( 1 , IVV ) . NE . 3 ) CALL RELAXY( IVV )
 1540       CONTINUE
C
            DO 1500 IENN = 1 , IJTRIG
            IE = JUE( IENN )
            CALL RECNC( IE , IDONE , ITL , ITR , JA , JB , JC , JD )
            CALL RECNC( JA , JADONE , ITL , ITR , JAA , JAB , JAC , JAD )
            CALL RECNC( JB , JBDONE , ITL , ITR , JBA , JBB , JBC , JBD )
            CALL RECNC( JC , JCDONE , ITL , ITR , JCA , JCB , JCC , JCD )
            CALL RECNC( JD , JDDONE , ITL , ITR , JDA , JDB , JDC , JDD )
 1500       CONTINUE
C
            DO 1510 IPRTCL = 1 , NPT
            ISM = IJKPRT( IPRTCL )
            DO 1510 KI = 1 , ITRIG
            JSM = NSCRSS( KI )
            IF( ISM . EQ . JSM ) IJKPRT( IPRTCL ) = NSMNPT
 1510       CONTINUE
C
C     UPDATE THE JSDELT ARRAY
C
            DO 1530 IS = 1 , ISDELT
            JSP = JSDELT( IS )
            DO 1530 KI = 1 , ITRIG
            JSM = ISCRSS( KI )
            IF( JSP . EQ . JSM ) JSDELT( IS ) = NSCRSS( KI )
 1530       CONTINUE
C
```

```
            INDCTR = 1
C
C --- EXIT POINT FROM SUBROUTINE -------------------------------------
C
C     ------
      RETURN
C     ------
C
C     ---
      END
```

APPENDIX C

COPIES OF PUBLICATIONS

AIAA

AIAA 89-2446

A REVIEW OF PROPULSION APPLICATIONS OF THE PULSED DETONATION ENGINE CONCEPT

S. EIDELMAN, W. GROSSMANN AND I. LOTTATI
SCIENCE APPLICATIONS INTERNATIONAL CORP.
APPLIED PHYSICS OPERATION
MCLEAN, VA

AIAA/ASME/SAE/ASEE
25th Joint Propulsion Conference
Monterey, CA / July 10-12, 1989

A REVIEW OF PROPULSION APPLICATIONS OF THE PULSED DETONATION ENGINE CONCEPT

S. Eidelman, W. Grossmann and I. Lottati

Applied Physics Operation
Science Applications International Corporation
1710 Goodridge Dr., McLean, VA 22102

Introduction

The early development leading to practical propulsion engines was almost completely associated with steady state engine concepts. Unsteady concepts, which initially appeared promising, never evolved from the conceptual state and have remained for the most part unexplored. The early work in unsteady propulsion suffered from a lack of appropriate analytical and design tools, a condition which seriously impeded the advancement of the unsteady concepts to a practical stage.

In this paper we review the historical development of unsteady propulsion by concentrating on one particular concept, the intermittent detonation engine, and discuss current research activities in this area. A review of the literature[1-24] reveals that a significant body of experimental and theoretical research exists in the area of unsteady propulsion. However, this research has not been extended to the point where a conclusive quantitative comparison can be made between impulsive engine concepts and steady state concepts. For example, the analysis given in References 8-11 of the performance of a detonation engine concept does not include frequency dependence, nor any analysis of losses due to multi-cycle operation. A new generation of analytical and computational tools exists today and allows us to revisit and analyse such issues with a high degree of confidence. Numerical simulation has developed to the state where it can now provide time dependent two and three dimensional modeling of complex internal flow processes[20,24,25] and will eventually result in tools for systematically analysing and optimising engineering design. In addition to a review of applications of the Pulsed Detonation Engine Concept here we will report results of a numerical study of the gasdynamics of a model of an air-breathing detonation engine with detailed analysis of the nonsteady flow pattern. This study was performed using new unsteady CFD tools which we will also describe.

Our paper is structured as follows: 1) historical review of the pulsed detonation development efforts; 2) description of the basic phenomenology of the air-breathing Pulsed Detonation Engine concept; 3) description of the mathematical formulation and new numerical scheme used to simulated the problem; 4) discussion of the simulation results; and 5) conclusions.

Historical Review

Constant Volume Combustion

From the very early development of jet-propulsion engines it was known that an engine based on a constant volume combustion process achieves higher efficiency than a constant pressure engine. This follows from a thermodynamic analysis of the engine cycle.[1]

Constant volume combustion was used in gas turbine engines at the beginning of this century, and the first gas turbine engines in commercial use were based on the constant volume cycle. Jet-propulsion engines were one of the applications of the constant volume cycle (or explosion cycle) which was explored in the late 1940s.[2] Although the explosion cycle operates at a larger pressure variation in the combustion chamber than in a pulse-jet[3,4], the cycle actually realized in these engines was not a fully constant volume one since the combustion chamber was open ended[2]. In Reference 2 the maximum pressure ratio measured in an explosion cycle engine was 3:1, whereas the pressure ratio for the same mixture under the assumption of a constant volume cycle would be 8:1. Also, this engine was limited by the available frequency of cycles, which in turn is limited by the reaction rate. A simple calculation [2] showed that if the combustion time could be reduced in this engine from 0.006 sec to 0.003 sec, the thrust per pound of mixture would increase 100%. Thus the explosion-cycle engine has two main disadvantages:

- Constrained volume combustion (as distinguished from constant volume combustion) does not take full advantage of the pressure rise characteristic of the constant volume combustion process.
- The frequency of the explosion cycle is limited by the reaction rate, which is only slightly higher than the deflagrative combustion rate.

The main advantage of the constant pressure cycle is that it leads to engine configurations with steady state

processes of injection of the fuel and oxidiser, combustion of the mixture, and expansion of the combustion products. These stages can be easily identified and the engine designer can optimise them on the basis of relatively simple steady state considerations.

At the same time an engine based on constant volume combustion will have an intermittent mode of operation, which may complicate its design and optimisation. We are interested in the question of whether this complication is worth the potential gains in engine efficiency.

Pulsed Detonation Engine as an Ultimate Constant Volume Combustion Concept

The detonation process, due to the very high rate of reaction, permits construction of a propulsion engine in which the constant volume process can be fully realised. In detonative combustion, the strong shock wave, which is part of the detonation wave, acts like a valve between the detonation products and the fresh charge. The speed of the detonation wave is about two orders of magnitude higher than the speed of a typical deflagration. This allows the design of propulsion engines with a very high power density. Usually, each detonation is initiated separately by a fully controlled ignition device, and the cycle frequency can be changed over a wide range of values. This also means that a device based on a detonative combustion cycle can be scaled and its operating parameters can be modified for a range of required output conditions. There have been numerous attempts to take advantage of detonative combustion for engine applications. In the following we give a description of the most relevant past experimental and analytical studies of the detonation engine concept.

Hoffmann's Report.

The first reported work on intermittent detonation is attributed to Hoffmann [5] in 1940. He operated an intermittent detonation test stand with acetylene-oxygen and benzine-oxygen mixtures. The addition of water vapor was used to prevent the highly sensitive acetylene-oxygen mixture from premature detonation. Hoffmann[5] indicated the importance of the spark plug location in reference to tube length and diffuser length. It was found that a continuous injection of the combustible mixture leads to only a narrow range of ignition frequencies which will produce an intermittent detonation cycle. These frequencies are governed by the time required for the mixture to reach the igniter, time of transition from deflagration to detonation, and time of expansion of the detonation products. Hoffmann attempted to find the optimum cycle frequency experimentally. It was discovered that detonation-tube firing occurred at lower frequencies than the spark-plug energising frequencies indicating that the injection flow rate and ignition were out of phase. Events prevented further work by Hoffmann and co-workers.

Nicholls Experiments.

A substantial effort in intermittent detonation engine research was done by a group headed by J. A. Nicholls[6-10] of The University of Michigan beginning in the early 50's. The most relevant work concerns a set of experiments carried out in a six foot long detonation tube[6]. The detonation tube was constructed from a one inch internal diameter stainless steel tube. The fuel and oxidiser were injected under pressure from the left end of the tube and ignited at the some distance down stream. The tube was mounted on a pendulum platform, suspended by support wires. Thrust for single detonations was measured by detecting tube (platform) movement relative to a stationary pointer. For multi-cycle detonations thrust measurement was achieved by mounting the thrust end of the tube to the free end of the cantilever beam. In addition to direct thrust measurements the temperature on the inner wall of the detonation tube was measured.

Fuel mixtures of hydrogen/oxygen, hydrogen/air, acetylene-oxygen and acetylene-air mixtures were used. The gaseous oxidiser and fuel were continuously injected at the closed end wall of the detonation tube and three fixed flow rates were used. Under these conditions the only parameters which could be varied were the fuel/oxidiser ratio and frequency of ignition. A maximum gross thrust of $\approx 3.2lb$ was measured in hydrogen/air mixture at the frequency of ≈ 30 detonations per second. The most promising results were demonstrated for the H_2/Air mixture, where a fuel specific impulse of $I_{sp} = 2100$ sec was reached. The maximum frequency of detonations obtained in all experiments was 35 Hz. The temperature measurements on the inner wall showed that for the highest frequency of detonations the temperature did not exceed 800° F.

In their later work,[8,9,10] the University of Michigan group concentrated on development of the Rotating Detonation Wave Rocket Motor. No further work on the pulsed detonation cycle was pursued.

Krzycki Experiments

In a setup somewhat similar to Nicholl's, L. J. Krzycki[11] performed an experimental investigation of intermittent detonations with frequencies up to 60 cps. An attempt was also made to analyze the basic phenomena using unsteady gas dynamic theory. Krzycki's attempt to analyse the basic phenomena relied on wave diagrams to trace characteristics, assumptions of isentropic flow for detonation and expansion, and incompressible flow for mixture injection processes. The most convincing

data from the experiments is the measurement of thrust for a range of initiation frequencies and mixture flow rates. Unfortunately no direct pressure measurement in the device are reported so that only indirect evidence exists of the nature of the process observed.

The basic test stand used by Krzycki is very similar to that used by Nicholls et al.[6] The length of the detonation tube and internal diameter were exactly the same as those in Nicholl's experiments. A Propane/Air mixture was continuously injected through a reversed-flow diffuser for better mixing, and ignited at the some distance from the injection point by an automobile spark plug. The spark frequency was varied from 1 to 60 cps. The spark plug power output was varied inversely with the initiation frequency and at the frequency of 60 cps was only 0.65 Joule. This fact alone eliminated the possibility of direct initiation of the detonation wave by the spark and consequently all of the experiments must have been based on transition from deflagration to detonation. According to experimental data and theory,[12] for direct initiation of a mixture of propane-air at the detonability limits, an energy release on the order of 10^6 Joules is required. Thus, the required deflagration-detonation transition region length would have been prohibitively large for the propane-air mixture. It follows that in all of the experiments a substantial part of the process was deflagrative. This resulted in low efficiency, and negligible thrust. Krzycki repeated the experiments of Nicholls using exactly the same size detonation tube and basically the same rates of injection of the detonable mixture. Krzycki's experimental results are very well documented, allowing a clear picture of the physical processes occurring in the tube to be deduced. A conclusion, arrived at by the author, was that thrust was possible from such a device but practical applications did not appear promising. It is unfortunate that, possibly based on Krzycki's extensive but misleading results, all experimental work related to the pulsed detonation engine concept stopped at this time.

Work Reported in Russian sources on Pulse Detonation Devices

A review of the Russian literature has not uncovered work concerning applications of pulsed detonation devices to propulsion. However there are numerous reports of applications of such devices for producing nitrogen oxide[13] (an old Zeldovich idea to bind nitrogen directly from air to produce fertilizers) and as rock crushing devices[14].

Korovin et al.[13] provide a most interesting account of the operation of a commercial detonation reactor. The main objective of this study was to examine the efficiency of thermal oxidation of nitrogen in an intermittent detonative process as well as an assessment of such technological issues as the fatigue of the reactor parts exposed to the intermittent detonation waves over a prolonged time. The reactor consisted of a tube with an inner diameter of 16 mm and length 1.3 m joined by a conical diffuser to a second tube with an inner diameter of 70 mm and length 3 m. The entire detonation reactor was submerged in running water. The detonation mixture was introduced at the end wall of the small tube. CH_4, O_2 and N_2 comprised the mixture composition and the mixture ratios were varied during the continuous operation of the reactor. The detonation wave velocity was measured directly by piezoelectric sensors placed in the small and large tubes. The detonation initiation frequency in the reactor was 2-16 Hz. It is reported that the apparatus operated without significant changes for 2000 hours.

Smirnov and Boichenko[14] studied intermittent detonations of gasoline-air mixtures in a 3 m long and 22 mm inner diameter tube operating in the 6-8 Hz ignition frequency range. The main motivation of this work was to improve the efficiency of a commercial rock crushing apparatus based on intermittent detonations of the gasoline-air mixtures.[15] The authors investigated the dependence of the length of the transitional region from deflagration to detonation on the initial temperature of the mixture.

As a result of the information contained in the Soviet reports, it can be concluded that reliable commercial devices based on intermittent detonations can be constructed and operated.

Development of the Blast Propulsion System at JPL

Work at the Jet Propulsion Laboratory (JPL) by Back, Varsi and others[16-19] concerned an experimental and theoretical study of the feasibility of a rocket truster using intermittent detonations of solid explosive useful for propulsion in dense or high-pressure atmospheres of certain solar system planets. The JPL work was directed at very specific applications; however, the studies[17-19] addressed some key issues of devices using unsteady process such as propulsion efficiency. The JPL studies have important implications to pulsed detonation propulsion systems.

Reference 19 gives the basic description of the test stand used. In this work a Deta sheet type C explosive was detonated inside a small detonation chamber attached to nozzles of various length and geometry. The nozzles, complete with firing plug, were mounted in a containment vessel which could be pressurized with the mixture of various inert gases from vacuum to 70 atm. The apparatus measured directly the thrust generated by single detonations of a small amount of solid explosive charge expanding into conical or straight nozzles.

Thrust and specific impulse was measured by a pendulum balance system.

Results obtained from an extensive experimental study of the explosively driven rocket have lead to the following conclusions. First, rockets with long nozzles show increasing specific impulse with increasing ambient pressure in CO_2 and N_2. Short nozzles, on the other hand, show that specific impulse is independent of ambient pressure. Most importantly, most of the experiments obtained a relatively high specific impulse of 250 seconds and larger. This result is all the more striking since the detonation of a solid explosive yields a relatively low energy release of approximately 1000 cal/gm compared with 3000 cal/gm obtained in hydrogen oxygen combustion. Thus, it can be concluded that the total losses in a thruster based on unsteady expansion are not prohibitive and, in principle, very efficient propulsion systems operating on intermittent detonations are possible.

Detonation Engine Studies at Naval Postgraduate School

A modest exploratory study of a propulsion device utilizing detonation phenomena was conducted at the Naval Postgraduate School.[20-23] During this study, several fundamentally new elements were introduced to the concept distinguishing the new device from previous ones.

First, it is important to note that the experimental apparatus constructed by Helman et al.[21] was the first successful self aspirating air breathing detonation device. Intermittent detonation frequencies of 25 Hz were obtained. This frequency was in phase with the fuel mixture injection through timed fuel valve opening and spark discharge. The feasibility of intermittent injection was established. Pressure measurements showed conclusively that a detonation process occurred at the frequency chosen for fuel injection. Further, self aspiration was shown to be effective. Finally, the effectiveness of a primary detonation as a driver for the main detonation was clearly demonstrated. Although the NPS studies were abbreviated, many of the technical issues considered to be essential for efficient intermittent detonation propulsion were addressed with positive results.

Simulations of Pulsed Detonation Engine Cycle at NASA-Ames Center

Recently Cambier and Adelman [24] carried out numerical simulations of a pulsed detonation engine cycle taking into account finite rate chemistry. Unfortunately, the simulations were restricted to a quasi-one dimensional model. The configuration considered had a 6 cm inner diameter 50 cm long main chamber which was attached to a 43 cm diverging nozzle. It was assumed that a stoichiometric mixture of hydrogen/air at 3.0 atmospheres is injected from an inlet on the closed end wall of the detonation chamber. At such conditions Cambier and Adelman estimated a large range of possible detonation frequencies of engine operation up to 667 Hz. The origin of this estimate is not clear from their work, since according to their simulations, the detonation, expansion and fresh charge fill requires 2.5 msec. This value leads to a maximum frequency of 400 Hz. The simulated engine performance yielded a large average thrust of 893 lb and an unusually high specific impulse of 6507 sec. These simulations were the first to demonstrate the use of modern CFD methods to address the technical issues associated with unsteady pulsed detonation concepts.

In the remaining sections we discuss a particular propulsion concept based on the results of the experiments of Helman et al.[22] and describe a computational study of its performance characteristics. The unsteady numerical scheme used for the study made use of unique simulation techniques; the key ingredients of these techniques are also described.

A Generic Pulsed Detonation Engine

The generic device we consider here is a small engine 15 cm long and 15 cm in diameter. The combustible gas mixture is injected at the closed end of the detonation chamber and a detonation wave propagates through the mixture. The size of the engine suggests a small payload, but the concept can be extended to larger payloads simply by scaling up the size of the detonation chamber and possibly combining a number of engines into one large propulsion engine. A key issue in the pulsed detonation engine concept is the design of the main detonation chamber. The detonation chamber geometry determines the propulsion efficiency and the duration of the cycle (frequency of detonations). Since the fresh charge for the generic engine is supplied from the external flow field, the efficiency of the engine depends on the interaction of the surrounding flow with the internal flow dynamics. The range of the physical processes requiring simulation in order to model the complex flow phenomena associated with the detonation engine performance is very broad. A partial list is:

1. Initiation and propagation of the detonation wave inside the chamber,
2. Expansion of the detonation products from the chamber into the air stream around the chamber at flight Mach numbers.
3. Reverse flow from the surrounding air into the chamber resulting from over expansion of the detonation products,
4. Pressure buildup in the chamber due to reverse flow. The flow pattern inside the chamber during post-exhaust pressure buildup determines the strategy

for mixing the next detonation charge,

5. Strong mutual interaction between the flow processes inside the chamber and flow around the engine.

All of these processes are interdependent and their timing is crucial to the engine efficiency. Thus, unlike simulations of steady state engines, the phenomena described above can not be evaluated independently.

The need to resolve the flow regime inside the chamber accounting for nozzles, air inlets etc., and at the same time resolve the flow around the engine, where the flow regime varies from high subsonic, locally transonic and supersonic, makes it a challenging computational problem.

The main issue is to determine the timing of the air intake for the fresh gas charge. It is sufficient to assume invicid flow for the purpose of simulating the expansion of the detonation products and fresh gas intake. In the following we present the first results of an invicid simulation of the detonation cycle in a cylindrical chamber. First, we describe our computational method for solving the time dependent Euler equations used in the study.

The Unsteady Euler Solver

A new second order algorithm for solving the Euler equations on an unstructured grid was used in our study of the detonation concept. The approach is based on first and second order Godunov methods. The method leads to an extremely efficient and fast Flow Solver which is fully vectorised and easily lends itself to parallelisation. The low memory requirements and speed of the method are due to the use of a unique data structure.

Until recently most CFD simulations were carried out with logically structured grids. Vectorisation and/or parallelisation did not present a problem. The increased need for simulation of flow phenomena in the vicinity of complex geometrical bodies and surfaces has led to the development of CFD codes for logically unstructured grids. The most successful of these unstructured grid codes are based on finite elements or finite volume methods. For an unstructured grid in two-dimensions, the computational domain is usually covered by triangles and the indices of the arrays containing the values of the hydrodynamic flow quantities are not related directly to the actual geometric location of a node. The calculations performed on unstructured grids evolve around the elemental grid shape (e.g. the triangle for two-dimensional problems) and there is no obvious pattern to the order in which the local integrations should be performed. Explicit integration of hydrodynamic problems on an unstructured grid requires that a logical substructure should be created which identifies the locations in the global arrays of all the local quantities necessary for the integration of one element. This usually results in a large price in computational efficiency, in memory requirements, and in code complexity. As a consequence, vectorisation for the conventional unstructured grid methods has concentrated on rearrangement of the data structure in a manner such that these locally centered data structures appear as global arrays. This can be done to some extent using machine dependent Gather-Scatter operations.[25,26] Additional optimisation can be achieved using localisation and search algorithms. However, these methods are complex and result in marginal improvement. Most optimised unstructured codes to date run considerably slower and require an order of magnitude more memory per grid cell then their structured counterparts. Parallelisation of the conventional unstructured codes is even more difficult, there is very little experience with unstructured codes on massively parallel computers.

The method we have developed overcomes these difficulties and results in code with speed and memory requirements comparable to those found in structured grid codes. Moreover, the ability to construct grids with arbitrary resolution leads to a flexibility in dealing with complex geometries not attainable with structured grids. The essence of the method is based on independent flux calculation across the edges of a dual baricentric grid, followed by node integration. This approach is order independent. Below we give the essential details of our algorithm; a complete description follows later.

Basic Integration Algorithm.

We begin by describing the first order Godunov method for the system of two-dimensional (axi-symmetric) Euler equations written in conservation law form as

$$\frac{\partial \vec{Q}}{\partial t} + \frac{\partial \vec{F}}{\partial x} + \frac{\partial \vec{G}}{\partial r} = -\frac{1}{r}\vec{C} \ , \qquad (1)$$

where,

$$\vec{Q} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ e \end{pmatrix}, \ \vec{F} = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ (e+p)u \end{pmatrix}, \ \vec{G} = \begin{pmatrix} \rho v \\ \rho v u \\ \rho v^2 + p \\ (e+p)v \end{pmatrix},$$

$$\vec{C} = \begin{pmatrix} \rho v \\ \rho v u \\ \rho v^2 \\ (e+p)v \end{pmatrix}.$$

Here u and v are the x and r velocity vector components, p is the pressure, ρ is the density and e is the total energy of the fluid per unit volume. It is assumed that a mixed (initial conditions, boundary conditions) problem is properly posed for the set of equations (1)

and that an initial distribution of the fluid parameters is given at $t = 0$ and some boundary conditions defining a unique solution are specified on the boundary of the computational domain.

We look for a solution of the system of equations represented by Eq. 1 in the computational domain covered by an unstructured grid. As an example, Fig. 1a shows the unstructured triangular grid used in the pulsed detonation engine simulation. Here most of the computational effort is committed to the resolution of the flow inside the engine detonation chamber and in the immediate vicinity of the nozzle. In Figure 1b an enlargement of the nozzle region is shown, illustrating the ability to represent geometry of arbitrary complexity and with localized resolution.

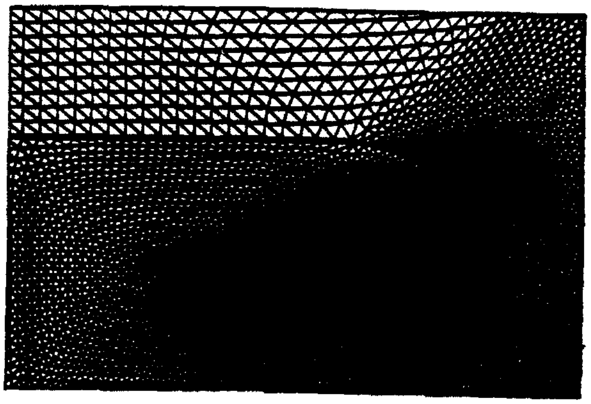


Figure 1a Computational domain and grid used in simulation of PDE operation.

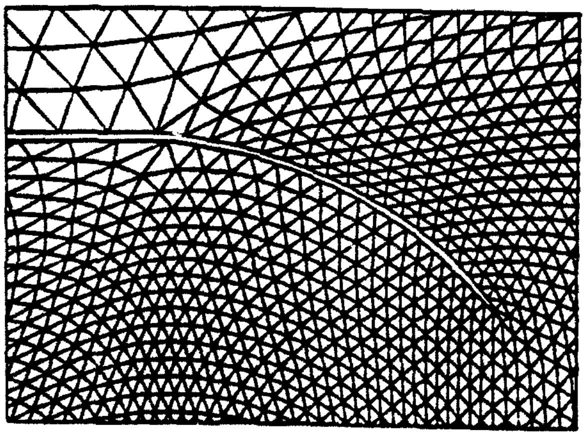


Figure 1b Enlargment of computational grid in the vicinity of the PDE nozzle.

Fig. 2 displays a fragment of the computational domain with the corresponding dual grid. The secondary or dual grid is formed by connecting the baricenters of the primary mesh, thus forming fi polygons around the primary vertices.

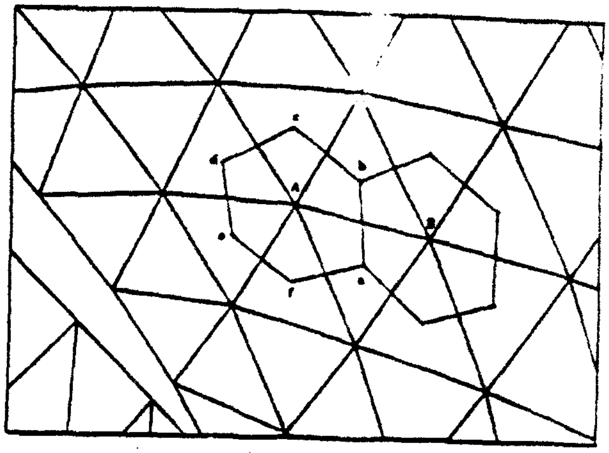


Figure 2 The primary (triangles) and secondary (poligons) unstructured grids.

We have found, as have others,[27] that the best practical representation of the integration volume is obtained when the dual grid is formed by connecting baricenters of the triangles. Integration by the Godunov method[28] can be divided into two basic steps: 1. Calculation of the fluxes at the edges of the secondary grid using solutions of a set of one dimensional Riemann problems; 2. Integration of the system of partial differential equations which amounts to addition of all the fluxes for every polygon at a particular time step.

To define the fluxes for the grid shown in Fig. 2 at every edge of the main grid it is necessary to solve the corresponding Riemann problem. For example, to define the flux at the edge ab, we solve the Riemann problem between points A and B. The solution of this problem is in coordinates local to the edge of the dual grid ab so that the tangential component of velocity will be directed along this edge (ab). Implementation of our approach requires maintaining strict consistency when defining the "left" and "right" states for the Riemann problems at the edges ab, bc, cd, de, ef, and fa. For this reason we define not only the location of the vertices and lengths of the edges but also the direction of the edges with respect to the primary grid. For the clockwise integration pattern in the same Polygon, point A will be the "right" state for all the Riemann problems related to this point and the neighbor will represent the "left" side of the diaphragm.

It is easy to see that the flux calculation is based on information at only two nodes and requires single geometrical parameters defining the edge of the secondary

grid that dissects the line connecting the two points. Thus, we can calculate all the values needed for flux calculation in one loop over all edges of the primary grid without any details related to the geometrical structures which these edges form. This in turn assures parallelisation or vectorisation of the algorithm for the bulk of the calculations involving the Riemann solver that provides the first order flux. The only procedure not readily parallelisable is the integration of the fluxes for the flow variables at the vertices of the grid. Here we use the "edge coloring" technique which allows us to split the flux addition loop into 7 or 8 loops for edges of different color. Each of these loops is usually large enough not to impair vectorisation. At this stage all the fluxes are added with their correct sign corresponding to the chosen direction of integration within the cell. The amount of calculation required here is minimal since the fluxes are known and need only to be multiplied at each time step by a simple factor and added to the vertex quantity.

Second Order Integration Algorithm

The second order solver is constructed along lines similar to that from the first-order method. At each cell edge the Riemann problem is solved for some specified pair of left and right conditions. The solution to this Riemann problem is then used in the calculation of fluxes which are added later to advance to the next integration step. The extension to second order is achieved by using extrapolation in space and time to obtain time-centered left and right limiting values as inputs for the Riemann problem. The basic implementation of the method of calculation of second order accurate fluxes is fundamentally the same as for one dimensional cases. The only difference is in the method of obtaining linear extrapolation of the flow variables as a first guess of their value at the edges of the dual grid. To obtain the first guess we need to know the gradient of some gasdynamical parameter U at the vertices of the primary mesh. The value of ∇ U can be evaluated by using a linear path integral along the edges which delineates the finite volume associated with the vertex. For vertex A in Figure 2 :

$$\int_A \nabla U dA = \oint_l U n \, dl \qquad (2)$$

where integration along the path l in this case is equivalent to integration along the edges ab, bc, cd, de, ef, fa. Knowing the gradient of the gasdynamic parameter in the volume related to vertex A will allow us to extrapolate the values of this parameter at any location within the volume. This permits us to evaluate the first guess for U at the edges of the dual grid. The final four steps of the implementation of the second order algorithm has been described previously.[28]

A schematic flow chart of the basic steps of the second order algorithm implementation is shown in Figure 3.

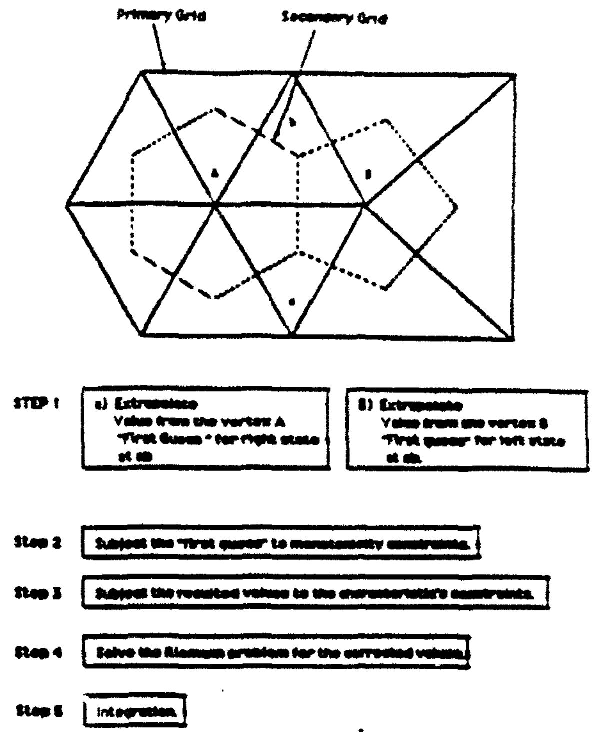


Figure 3 Grid schematic and outline of steps for second order Godunov method.

Simulations of the Generic Pulsed Detonation Engine

In this section we present sample results of simulations of the generic PDE device using the numerical code described in the preceding section. In Figure 1a the computational domain containing the PDE main detonation chamber is shown covered with the unstructured grid. In our sample simulation we have chosen a small ≈ 15 cm long and ≈ 15 cm internal diameter cylindrical chamber with a small converging nozzle. This geometry is one of a number of the geometries we have analyzed in a parametric study whose goal is to evaluate and optimise a typical PDE device. The device shown in Figure 1a does not represent the optimum and is given here to illustrate our methodology. We consider a situation when the PDE serves as a main thruster for a vehicle traveling in air with the velocity of M = 0.9 and located at the aft end of the vehicle. The main objectives of the simulations presented here are:

1. To find the maximum cycle frequency. This is determined by the time required from detonation, exhaust of combustion products and intake of fresh charge for the next detonation.

2. To calculate the thrust produced during each cycle and the integrated thrust as a function of time.

The simulation begins at $t = 0$ when a strong detonation wave is initiated inside the detonation chamber. Initially the detonation wave travels from the open aft end of the chamber towards the interior with a maximum velocity of 1800 $\frac{m}{sec}$ and maximum pressure of $20 * 10^5 Pa$. The distribution of pressure, velocity, and density of the detonation wave is defined through the *selfsimilar* solution for a planar detonation wave. The wave was directed towards the interior of the chamber to capture the kinetic energy of the wave and to prolong exposure of the inner chamber walls to the high pressure. In Figure 4a simulation results are shown at time $t = 0.19$ *msec* in the form of pressure contours and particle paths from different locations inside and outside the detonation chamber. From the pressure contour plots we observe that the shock reflection from the inner wall has taken place and detonation products are expanding into the ambient airstream. The flow inside the chamber is choked due to the converging nozzle and the maximum pressure behind the shock is $\approx 8atm$. The pressure inside the chamber is less than 3 atm. The strong expansion of the detonation products into the ambient airstream produces a shock wave with a spherical like front rapidly decaying in strength. As a result of the interaction of the expanding detonation products with the external flow a large toroidal vortex is created. The vortex is carried away quickly from the chamber by the external flow and by its own flow momentum.

In Figure 4a we also show particle paths for the particles introduced inside the chamber and outside just above the nozzle. Examination of these trajectories allows us to follow the dynamics of the chamber evacuation and refill. In order to track the detonation products we initially place marker particles inside the chamber at three cross sections in clusters of four distributed equally normal to the detonation chamber axis. Each particle has a different color; however, particles in the same cluster have the same shade of color. At the three chosen cross sections we have designated shades of red, yellow, and blue for the particles located correspondingly at the left end, center and beginning of the nozzle cross sections of the chamber. The movement of these particles is shown by connecting them with a continuous line beginning with particle location at $t = 0$ to the present time. In Figure 4a we observe that at time $t = 0.19 * 10^{-3}$ sec all particles originally in the nozzle cross section and three of the particles originally in the mid section have left the detonation chamber. However, particles originally introduced on the inner wall of the chamber have only advanced to the nozzle region.

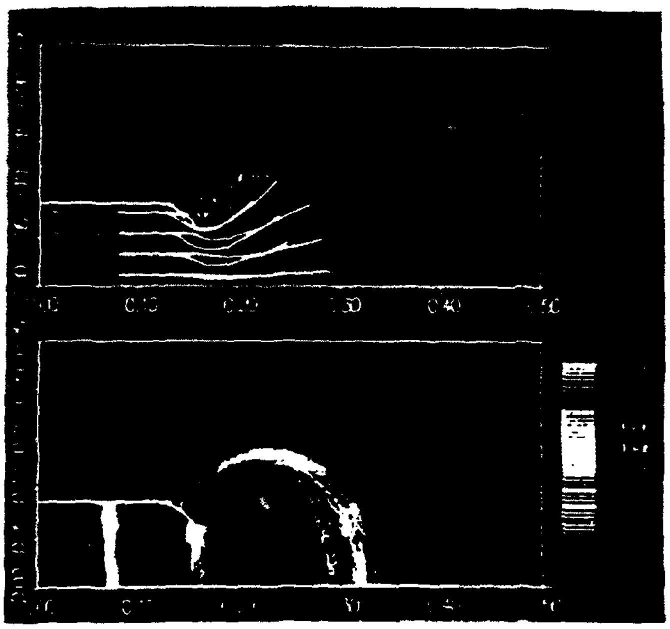
We use a different technique for observing the motion of the ambient gas outside the chamber. Here a cluster of seven particles is introduced every $0.5 * 10^{-4}$ seconds in the external flow above the nozzle. All such particles are traced as they move with the flow until they leave the computational domain. At any given time only the current location of the particle is displayed, and since the particles are introduced periodically with time there is a large number of particles to trace. We assign a color to every cluster of external particles to keep track of the time when they were introduced in the calculation. The colors vary from magenta for those particles introduced early in calculation, to blue for those introduced shortly at the time before the end of a detonation cycle. In Figure 4a corresponding to very early times, only one cluster of external particles is visible. This cluster was introduced at $t = 0$ and is tracking the expanding flow of the detonation products.

In Figure 4b the simulation results are shown for $t = 1.7 * 10^{-3}$ sec. The pressure contours show that a shock wave develops at the external edge of the nozzle as a result of a strong expansion of the Mach 0.9 external flow. A result of overexpansion of the detonation products is that the pressure inside the detonation chamber is lower than the ambient pressure, causing the shock to be located lower on the external surface of the nozzle. The external flow about the chamber has a stagnation point on the axis of symmetry downstream at $\approx 25cm$. At this time as it is evident from the particle trajectories that most of the detonation products have left the chamber. Figure 4b shows one continuous trace of the particles originating at the back wall of the detonation chamber having advanced well ahead of the stagnation point in the external flow.

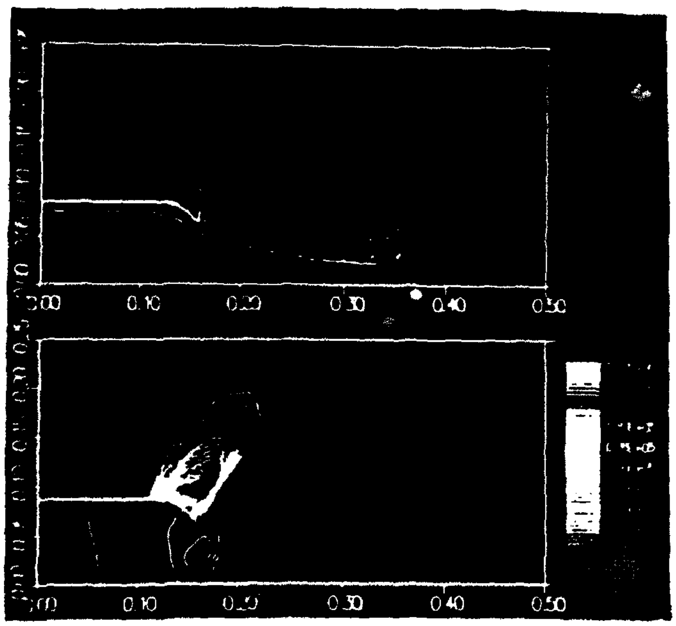
The marker particles released outside and just above the nozzles exit show two distinct flow paths. One path takes the flow past the stagnation point to the right of the detonation chamber; this flow path is marked by the four upper particle traces. Another flow path, marked by three lower particle released close to the nozzle surface is deflected towards the detonation chamber exit. Figure 4b shows this deflected stream approaching the detonation chamber nozzle. The magenta color of these particles indicates they were released at $\approx 0.5 * 10^{-3}$ sec.

Figure 4c corresponds to the simulation time $t = 0.47 * 10^{-3}$ sec. The pressure inside the chamber has risen $\approx 1atm$. Higher pressure at the chamber exit has caused the shock standing on the external surface of the nozzle to move upwards. The particles marking the movement of fresh air into the chamber show these to be well inside with some reflecting from the end wall giving a second stagnation point for the reversed fresh airflow.

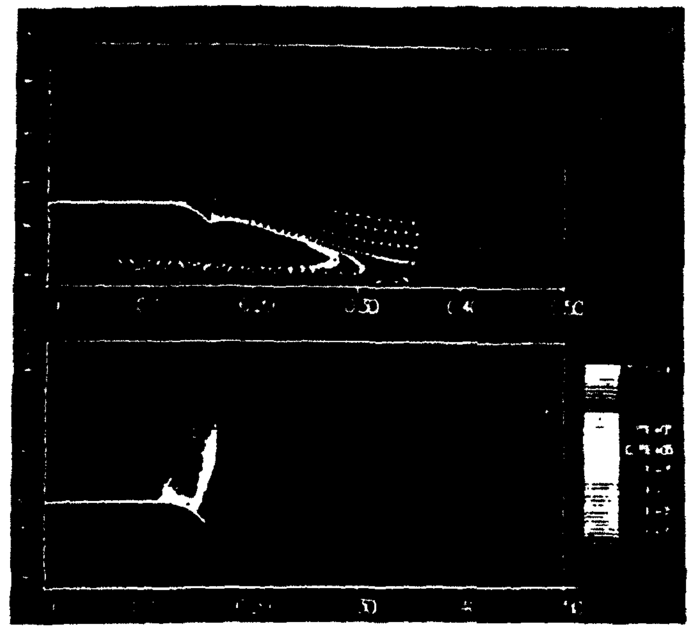
Figure 4d corresponds to the end of the first cycle when the detonation chamber should be filled with fresh charge and ready for the next detonation. In this figure the particle paths indicate that the chamber refills in a



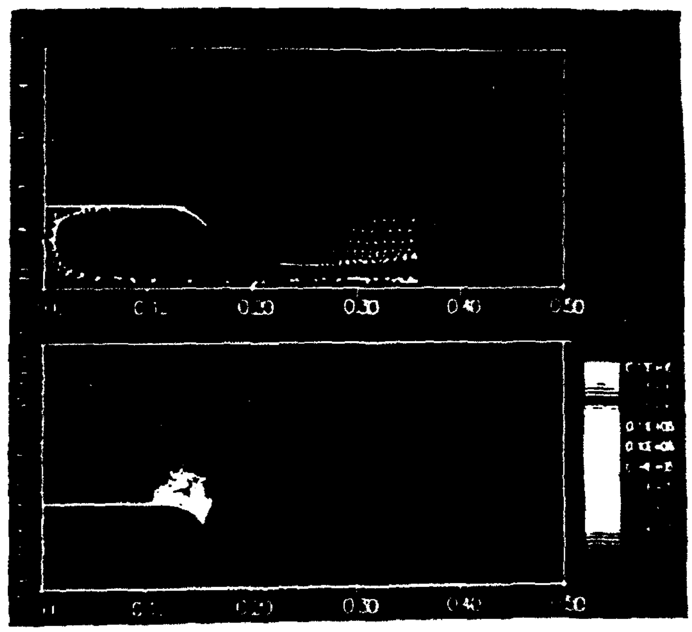
a) t = 0.19 msec,



b) t = 1.7 msec,



c) t = 4.7 msec,



d) t = 7.4 msec, end of first detonation cycle.

Figure 4 Pressure contours and particle paths for various times during the PDE simulation;
a) t = 0.19 msec, b) t = 1.7 msec,
c) t = 4.7 msec, d) t = 7.4 msec, end of first detonation cycle.

pattern suitable for fast mixing of the fuel-air mixture. We conjecture that fuel injection along the chamber axis will promote fast fuel-air mixing. We can see in Figure 4d that the farther injection of the external air flow inside the chamber stopped, and from that point on the mixture composition in the chamber will be fixed.

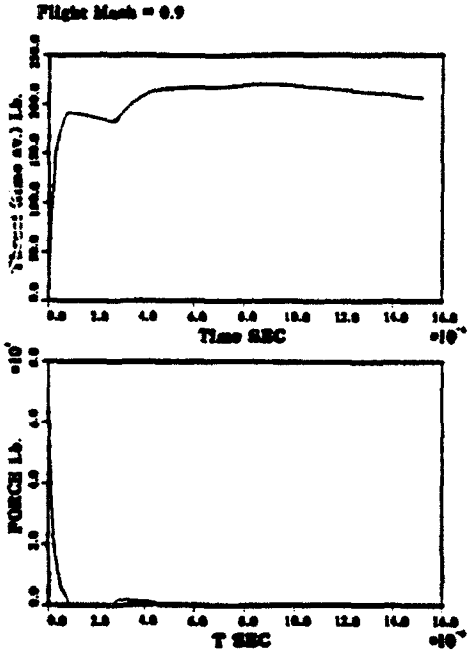


Figure 5 Thrust and force generated by PDE as function of time.

In Figure 5 total force and time averaged thrust generated by the device in the simulations discussed previously are shown as a function of time. The time averaged thrust is based on the total time for one cycle. As seen in Figure 5, initially a very large force of $\approx 7 * 10^4 lb$ is felt on the end wall of the detonation chamber. This is a result of the inwardly moving detonation wave used in our simulation. Very early during the sequence, this wave reflects from the left wall of the detonation chamber generating briefly a large force. This force rapidly decays and at $t \approx 1.0 * 10^{-4}$ sec changes sign due to interaction of the strong shock wave with the converging nozzle. This effect is noticeable in the thrust data; the average thrust decreases somewhat after reaching levels of $\approx 200 lbs$. The shock partially reflects from the converging nozzle walls and generates a wave moving to the left wall. The reflected wave thereafter generates positive thrust from $t \approx 3.0 * 10^{-4}$ sec. Finally thrust levels reach the maximum of 225 lbs. and then decays slowly as a result of the cross sectional drag force. The simulations predict that to sustain this level of thrust will require a detonation frequency of about 150 Hz.

Conclusions

The main intent of the present study was to carry out a review of the relevant literature in the area of detonation propulsion, to assess the state-of-the-art, and to recommend future research based on our findings. We have reviewed the literature and presented our summary in first section of this paper. Our initial conclusion from the review is that there is a substantial body of evidence leading toward the possibility of producing propulsion engines with significant thrust levels based on an intermittent detonation.

Most of the historical attempts at producing thrust based on the intermittent detonation cycle were carried out with the same basic experimental setup; namely, a long straight detonation tube employing forced fuel injection at the closed tube end. We have discussed the many reasons why such a device cannot take proper advantage of the physical processes associated with detonation.

The experiments performed at the Naval Postgraduate School using a self-aspirating mode of operation for pulsed detonation thruster produced very useful results which, upon further examination, provide us with a route towards practical propulsion engines of variable thrust levels which are both controllable and scalable.

We have explored some of the implications of the possible applications of the self aspirating detonation engine concept and have developed a suitable numerical simulation code to be used as a design, analysis and evaluation tool. In fact, the preliminary analysis of a candidate detonation chamber flow properties was shown to be dominated completely by unsteady gasdynamics. An attempt to understand the flow properties based on any steady state model or one-dimensional unsteady analytical model will miss such important aspects as fuel-air mixing and, shock refielction from internal geometrical obstacle such as the converging nozzle. The unsteady simlation code developed during the course of our study is a necessary tool that we plan to use in a study leading to a feasible prototype engine design realizing the full potential of the intermittent detonation process.

Acknowledgements

The authors would express appreciation to Drs. Adam Drobot and Aharon Friedman for helpful suggestions and advice. The work reported on here was supported by DARPA under contract N66001-88-D-0088.

REFERENCES

1. Stodola, A., Steam and Gas Turbines, McGraw-Hill Inc., 1927.
2. Zipkin, M. A., and Lewis G. W., "Analytical and Experimental Performance of an Explosion-Cycle Combustion Chamber of a Jet Propulsion Engine," NACA TN-1702, Sept. 1948.
3. Shultz-Grunow, F., "Gas-Dynamic Investigation of the Pulse-Jet Tube," NACA TM-1131, Feb. 1947.
4. Zinn, B. T., Miller, N. Carvelho, J. A., and Daniel B. R.,"Pulsating Combustion of Coal in Rijke Type Combustor," 19th International Symposium on Combustion, 1197-1203, 1982.
5. Hoffmann, N., "Reaction Propulsion by Intermittent Detonative Combustion," Ministry of Supply, Volkenrode Translation, 1940.
6. Nicholls, J. A., Wilkinson, H. R. and Morrison, R. B. "Intermittent Detonation as a Thrust-Producing Mechanism," Jet Propulsion, 27, 534-541, 1957.
7. Dunlap, R., Brehm, R. L. and Nicholls, J. A., " A Preliminary Study of the Application of Steady State Detonative Combustion of a Reaction Engine", ARS J., 28, 451-456, 1958.
8. Nicholls, J. A., Gullen, R. E. and Ragland K. W., "Feasibility Studies of a Rotating Detonation Wave Rocket Motor," Journal of Spacecrafts and Rockets, 3, 893-898, 1966.
9. Adamson, T. C. and Olsson, G. R., "Performance Analysis of a Rotating Detonation Wave Rocket Engine," Astronautica Acta, 13, 405-415, 1967.
10. Shen, P. I., and Adamson, T. C., "Theoretical Analysis of a Rotating Two-Phase Detonation in Liquid Rocket Motors," Astronautica Acta, 17, 715-728, 1972.
11. Krzycki, L. J., Performance Characteristics of an Intermittent Detonation Device, Navweps Report 7655, U. S. Naval Ordnance Test Station, China Lake, California 1962.
12. Matsui, H., and Lee, J. H., " On the Measure of the Relative Detonation Hazards of Gaseous Fuel-Oxygen and Air Mixtures," Seventeenth Symposium (International) on Combustion, 1269-1280, 1978.
13. Korovin, L. N., Losev A., S. G. Ruban and Smekhov, G. D. "Combustion of Natural Gas in a Commercial Detonation Reactor," Fiz. Gor. Vzryva, Vol. 17, No.3, p.86, 1981.
14. Smirnov, N. N., Boichenko, A. P., " Transition from Deflagration to Detonation in Gasoline-Air Mixtures," Fiz. Gor. Vzryva, 22, No.2, 65-67, 1986.
15. Lobanov, D. P., Fonbershtein, E. G., Ekomasov, S. P., "Detonation of Gasoline-Air Mixtures in Small Diameter Tubes," Fiz. Gor. Vzryva, 12, No.3, 446, 1976.
16. Back, L. H., "Application of Blast Wave Theory to Explosive Propulsion," Acta Astronautica, 2, No 5/6 , 391-407, 1975.
17. Varsi, G., Back, L. H., and Kim, K., "Blast Wave in a Nozzle for Propulsion Applications," Acta Astronautica, 3, 141- 156, 1976.
18. Kim, K., Varsi, G., Back and L. H., "Blast Wave Analysis for Detonation Propulsion," AIAA Journal, Vol. 10, Oct. 1977.
19. Back L. H., Dowler, W. L. and Varsi, G., "Detonation Propulsion Experiments and Theory," AIAA Journal Vol. 21 Oct. 1983.
20. Eidelman, S., Shreeve, R. P., "Numerical Modeling of the Nonsteady Thrust Produced by Intermittent Pressure Rise in a Diverging Channel," ASME FED-Vol. 18, Multi-Dimensional Fluid Transient, p.77, 1984.
21. Eidelman, S., "Rotary Detonation Engine," U.S. Patent 4 741 154, 1988.
22. Helman, D., Shreeve, R. P., and Eidelman, S., "Detonation Pulse Engine,", AIAA-86-1683, 24[nd] Joint Propulsion Conference, Huntsville, 1986.
23. Monks, S. A., "Preliminary Assessment of a Rotary Detonation Engine Concept," MSc Thesis, Naval Postgraduate School, Monterey, California, Sept. 1983.
24. Camblier, T. L. and Adelman, N. G., "Preliminary Numerical Simulations of a Pulsed Detonation Wave Engine," AIAA-88-2960, AIAA 29th Joint Propulsion Conference, Boston 1988.
25. R. Lohner, K. Morgan, and D.C. Zienkiewics, "Finite Element Methods for High Speed Flows," *AIAA 7th Computational Fluid Dynamics Conference*, Cincinnati, Ohio, AIAA Paper 85-1531 (1985).
26. R. Lohner and K. Morgan, "Improved Adaptive Refinement Strategies for Finite Element Aerodynamic Computations," *AIAA 29th Aerospace Sciences Meeting*, Reno, Nevada, AIAA Paper 86-0499 (1986).
27. T.J. Barth and D.C. Jespersen, "The Design and Application of Upwind Schemes on Unstructured Meshes," 27[th] *Aerospace Sciences Meeting*, AIAA-89-0366, Reno, Nevada.
28. S. Eidelman, P. Collela, and R.P. Shreeve, " Application of the Godunov Method and It's Second Order Extension to Cascade Flow Modeling," *AIAA Journal*, v. 22, 10, 1984.

AIAA

AIAA-90-0460

COMPUTATIONAL ANALYSIS OF PULSED DETONATION ENGINES AND APPLICATIONS

S. EIDELMAN, W. GROSSMANN
AND I. LOTTATI
SCIENCE APPLICATIONS
INTERNATIONAL CORPORATION
McLEAN, VA

28th Aerospace Sciences Meeting

January 8-11, 1990/Reno, Nevada

COMPUTATIONAL ANALYSIS OF PULSED DETONATION ENGINES AND APPLICATIONS

S. Eidelman, W. Grossmann and I. Lottati
Applied Physics Operation
Science Applications International Corporation
1710 Goodridge Drive
McLean, Virginia 22102

1. Introduction

This paper presents the results of a computational fluid dynamic simulation/ parameter study of the SAIC Pulsed Detonation Engine (PDE) concept. Results from computer simulations of generic PDE geometries over a wide range of subsonic and supersonic flight Mach numbers indicate that potentially practical detonation engines can now be conceptualized and optimized for specific flight requirements and missions. Specifically, the study shows that primary propulsion for aerodynamic vehicles of the PENAID variety may be possible at Mach numbers $0.5 < M < 0.8$, thrust levels on the order of 100 pounds and a specific fuel consumption of the order of 1 lb./(lb.hr.). The predicted performance places the PDE propulsion concept in a strongly competitive position compared with present day small turbojets. The PDE concept has the added attractiveness of rapid variable thrust control, no moving parts and the potential for low cost manufacturing. Finally, the PDE concept is scalable over a wide range of engine sizes and thrust levels. For example, it is theoretically possible to produce PDE engines on the order of one to several inches in diameter and thrusts on the order of pounds, as well as devices which provide thousands of pounds thrust.

A literature search of past research on related concepts and devices uncovered important information which proved useful in pursuing our present study. A review of the literature[1-24] reveals that a significant body of experimental and theoretical research exists in the area of unsteady propulsion. However, this research was not sufficiently extensive to provide a conclusive quantitative comparison between impulsive engine concepts and steady state concepts. In addition, the computational and analytical techniques were not sufficiently developed in the past to treat the inherently unsteady flows of pulsed engines. A new generation of analytical and computational tools exists today, allowing us to revisit and analyze these devices with a high degree of confidence.

Our paper is organized into the following sections: 2) description of the basic phenomenology of the air-breathing Pulsed Detonation Engine concept; 3) discussion of the numerical simulation results; and 4) conclusions. Details of the mathematical formulation of the simulation and a discussion of the numerical code used in the present study are given elsewhere.[25,26]

2. The Generic Pulsed Detonation Engine

A detonation process, due to the very high rate of reaction, leads to a propulsion concept in which the constant volume process can be fully realized. In detonative combustion, the strong shock wave, which is part of the detonation wave, acts like a valve between the detonation products and fresh charge. The speed of the detonation wave is about two orders of magnitude higher than the speed of a typical deflagration. This allows the design of propulsion engines with a very high power density. Each detonation has to be initiated separately by a fully controlled ignition device, with a wide range of variable cycle frequencies. There is no theoretical restriction on the range of operating frequencies; they are uncoupled from acoustical chamber resonancies. This is very important feature of the constant volume detonation process that differentiates it from the process occurring in a pulse-jet;[3-4] the pulse jet cycle is tuned to the acoustical resonances of the combustion chamber. This leads to a lack of scalability for the pulse jet concept.

A physical restriction dictating the range of detonation frequency arises from the rate at which the fuel/air mixture can be introduced into the detonation chamber. This also means that a device based on a detonative combustion cycle can be scaled and its operating parameters can be modified for a range of required output conditions. There have been numerous attempts to take advantage of detonative combustion for engine applications. The most recent and successful of these attempts was carried out at the Naval Postgraduate School (NPS) by Helman et al.[22] During this study, several fundamentally new elements were introduced to the concept distinguishing the NPS research device from previous studies. First, it is important to note that the NPS experimental apparatus was the first successful self aspirating air breathing detonation device. Intermittent detonation frequencies of 25 Hz were obtained. This frequency

was in phase with the fuel mixture injection through timed fuel valve opening and spark ignition. The feasibility of intermittent injection was established. Pressure measurements showed conclusively that a detonation process occurred at the frequency chosen for fuel injection. Further, self aspiration was shown to be effective. Finally, the effectiveness of a primary detonation as a driver for the main detonation was clearly demonstrated. Although the NPS studies were abbreviated, many of the technical issues considered to be essential for efficient intermittent detonation propulsion were addressed with positive results.

As a result of the survey of past research on intermittent detonation devices, we have focussed our attention on the NPS experiments of Helman et al.[22] The remainder of this paper is concerned with a computer simulation of performance characteristics of such a device. We have chosen a generic geometry, applicable to certain present day vehicle and mission requirements, and have parametrically varied key features which affect performance and assessed the effects of these variations.

The generic device we consider here is a small engine 15 cm long and 15 cm in diameter. Figure 1 shows a schematic of the basic detonation chamber attached to the aft end of a generic aerodynamic vehicle. The combustible gas mixture is injected at the closed end of the detonation chamber and a detonation wave propagates through the mixture. The size of the engine suggests a small payload or aerodynamic vehicle, but the concept can be extended to larger payloads simply by scaling up the size of the detonation chamber and possibly combining a number of chambers into one larger engine. As an example, a PENAID vehicle has been conceptualized requiring an engine with a diameter of roughly 15 cm and a useful continuous thrust at Mach 0.8 approximately 60-90 pounds. Such an engine should have a specific fuel consumption in the range of 1.7 to 1.9 lb. fuel/hr per pound and an endurance on the order of 10-30 minutes. These specifications are met by present day small turbojets. Hence, in order to be competitive, a PDE must at least meet these requirements. Should this prove to be the case, a PDE with no moving parts would be a very attractive engine from the point of view of performance, ease of manufacturing and cost.

A key issue in the pulsed detonation engine concept is the design of the main detonation chamber. The detonation chamber geometry determines the propulsion efficiency and the duration of the cycle (frequency of detonations). Since the fresh charge for the generic engine is supplied from the external flow field, the efficiency of the engine depends on the interaction of the surrounding flow with the internal flow dynamics. The range of the physical process requiring simulation in order to model the complex flow phenomena associated with the detonation engine performance is very broad. A partial list is:

1. Initiation and propagation of the detonation wave inside the chamber,
2. Expansion of the detonation products from the chamber into the air stream around the chamber at flight Mach numbers,
3. Fresh air intake from the surrounding air into the chamber.
4. The flow pattern inside the chamber during post-exhaust pressure buildup which determines the strategy for mixing the next detonation charge,
5. Strong mutual interaction between the flow inside the chamber and surrounding the engine.

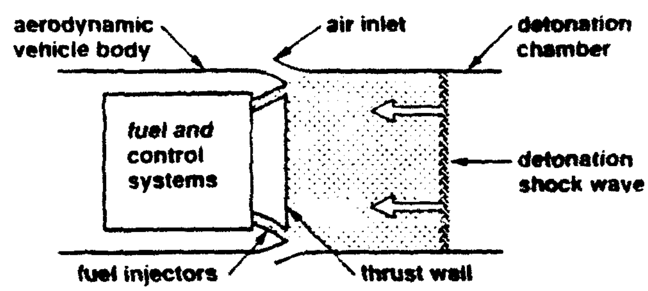


Figure 1. Schematic of the generic PDE showing detonation chamber, inlet, detonation wave, fuel injectors and position relative to an aerodynamic vehicle.

All of these processes are interdependent, and interaction and timing are crucial to engine efficiency. Thus, unlike simulations of steady state engines, the phenomena described above can not be evaluated independently.

The need to resolve the flow regime inside the chamber accounting for nozzles, air inlets etc., and at the same time resolve the flow outside and surrounding the engine, where the flow regime varies from high subsonic, locally transonic and supersonic, makes it a challenging computational problem.

The single most important issue is to determine the timing of the air intake for the fresh charge leading to repetitive detonations. It is sufficient to assume inviscid flow for the purpose of simulating the expansion of the detonation products and fresh air intake. The assumption of inviscid flow makes the task of numerically simulating the PDE flow phenomena somewhat easier than if a fully viscous flow model were employed. For the size of the generic device studied in this work the effects of viscous boundary layers are negligible with the exception of possible boundary layer effects on the valve and inlet geometries discussed subsequently. Boundary layer effects on the present results are discussed later.

3. Results of Simulational Parameter Study

As mentioned, and as shown in Figure 1, we have chosen a small $\approx$ 15 cm long and $\approx$ 15 cm internal diameter cylindrical chamber as the basic device. In the figure, the detonation chamber including detonation wave and inlet valves are shown schematically. The PDE, as envisioned in the present study, would most naturally be applicable for a class of aerodynamic vehicles such as target drones and PENAID missiles among others. The exact details of this basic chamber geometry were modified during the course of the study in order to obtain the required aerodynamic or propulsion effects; however, these modifications did not significantly change the total internal volume of the chamber. Thus, the performance results for the different cases can be compared holding the chamber volume constant. The schematic shown in Figure 1 does not represent an optimum configuration and is given here mainly to illustrate our methodology. We consider a situation where the PDE serves as the main thruster for an aerodynamic vehicle traveling in air with Mach numbers between M = 0.2 and M = 5.0, and is located at the aft end of the vehicle. The main objectives of our study are:

1. To calculate the thrust produced during each cycle and the integrated thrust as a function of time,
2. To find the maximum cycle frequency. This is determined by the time required from detonation to the final exhaust of combustion products and intake of fresh charge for the next detonation,
3. To evaluate the parametric dependence of the thrust and detonation frequency on the flight Mach number and detonation chamber geometry details.

In addition to the technical objectives outlined above, we have set another goal for our study. We require that the best, but by no means optimum, configuration produce a minimum of 60 pounds thrust at an operating frequency of 140 cycles per second. The definition of best is that configuration which satisfies the technical objectives outlined above and meets the operational goal of 60 pounds thrust and 140 Hz frequency over the flight regimes from M=.2 to M=0.9. (The Mach number range corresponds to that of a PENAID missile mission profile.)

To achieve these objectives we have conducted a comprehensive parametric simulational study of the PDE performance. We have studied PDE engine performance for a range of Mach numbers with two separate initial detonation locations in the chamber and for various geometry modifications. In addition to the range of subsonic Mach numbers we have examined PDE performance in the supersonic regime, 2 < M < 5. The geometry modifications included converging exhaust nozzles, inlets and dynamic valves. A computer simulation code was developed and optimized for a Stellar graphics workstation to carry out the analysis. In addition, a particle-tracing package was developed and implemented in the code. This allowed us to analyze the flow pattern inside and outside the detonation chamber, the main sources creating this pattern as a function of time, and the composition of the resulting gas mixture (air/detonation products).

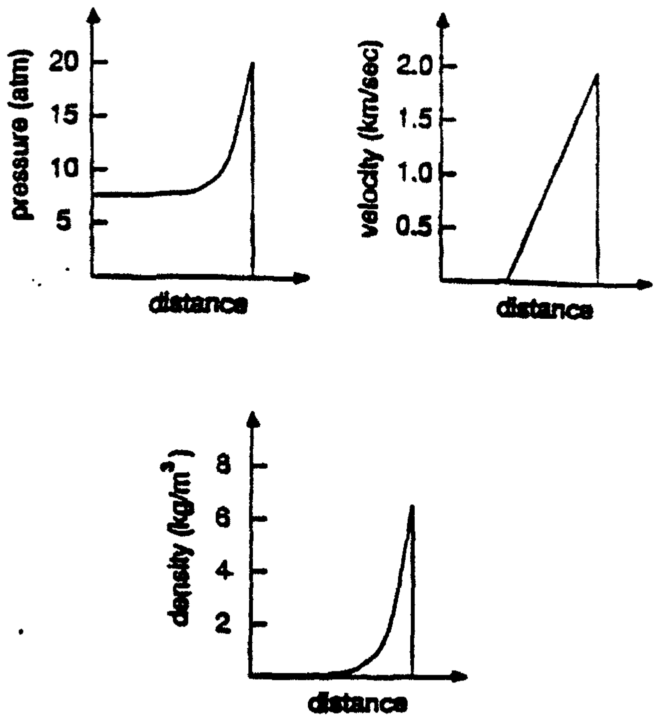


Figure 2. Distribution of gasdynamic parameters behind the detonation wave according to a 1-D self-similar solution.

First we will describe in detail the results for a typical simulation, Case 1, and illustrate the main features of our analysis.

<u>Case 1</u>. The simulation begins at $t = 0$ when a strong detonation wave is initiated inside the detonation chamber. The detonation chamber for this case includes a simple annular inlet which remains open during operation. The external freestream Mach number is 0.8. The specific fuel chosen for the present simulations is ethylene. The chemical reaction occurring in the ethylene/air detonation process is given by:

$$C_2H_4 + 3O_2 + 11.24N_2 \longrightarrow 2H_2O + 2CO_2 + 11.24N_2$$

The detonability limits of ethylene in air range from 4 to 12% by volume and depend somewhat on temperature and pressure. We assume for the sake of simplicity that

the fuel/air ratio is 6% by volume. Because the detonation initiation and propagation (detonative combustion) takes place several orders of magnitude faster than any of the other flow processes in or surrounding the device, finite rate chemistry is not included in the simulations. Instead the equation of state for the flow in the chamber immediately after detonative combustion was adjusted to represent the correct physical state of the combustion products. Initially the detonation wave travels from the closed end of the chamber towards the open aft end with a maximum velocity of 1800 m/sec and maximum pressure of $20 * 10^5 Pa$. The initiation of the main detonation wave is assumed to take place via a device proposed and successfully implemented by Helman, *et al.*;[22] namely, a primary detonation is established in a small tube containing an oxygen rich mixture. This mixture requires a low initiation energy but will sustain a detonation which, in turn, is used to trigger the main detonation wave. We do not model the detonation tube; but, we assume that such a device is present to trigger the main detonation at $t = 0$. The distribution of pressure, velocity, and density of the detonation wave is defined through the selfsimilar solution for a planar detonation wave. A schematic of the detonation wave distribution in space in Figure 2 shows pressure, temperature and velocity as a function of the spatial extent of the detonation wave.

In Figure 3a simulation results are shown at time $t = 0.64$ *m/sec* in the form of pressure contours and particle paths from different locations inside and outside the detonation chamber. The free stream Mach Number is 0.8. From the pressure contour plots we observe that the detonation shock wave has left the chamber and is freely expanding outwardly in the external flow. The strong expansion of the detonation products into the ambient airstream produces a shock wave with a spherical-like front that rapidly decays in strength away from the source. A large toroidal vortex is created as a result of the interaction of the expanding detonation products with the external flow. The vortex is carried away quickly from the chamber by the external flow and by its own momentum. At the time shown in Figure 3a, the detonation products are almost fully expanded into the ambient air and the maximum pressure at the front of the shock wave is 1.2 atm. As a result of this expansion the detonation products inside the detonation chamber are overexpanded and their pressure is *0.45 atm*.

In the upper frame of Figure 3a we show particle paths for the marker particles introduced inside the chamber and outside just above the nozzle exit. Examination of these trajectories allows us to follow the dynamics of the chamber evacuation and subsequent refill. In order to track the detonation products we initially place marker particles inside the chamber at three separate cross sections in clusters of four. Each particle has a different color; however, particles in the same cluster have the same shade of color. At the three chosen cross sections we have designated shades of red, yellow, and blue for the particles located correspondingly at the left chamber end, center and aft end of the nozzle cross section. The movement of these particles is shown by connecting them with a continuous line beginning with particle location at $t = 0$ to the present time. In Figure 3a we observe that at time $t = 0.64 * 10^{-3}$ sec all particles originally in the nozzle cross section (the cross section at the aft end) and three of the particles originally in the mid section have left the detonation chamber. However, particles originally introduced at the inner end wall of the chamber (red traces) have only advanced to the nozzle region.

We use a different particle technique for observing the motion of the ambient gas outside the chamber. Here a cluster of seven particles is introduced every $0.5 * 10^{-4}$ seconds in the external flow above the nozzle. All such particles are traced as they move with the flow until they leave the computational domain. At any given time only the current location of the particle is displayed, and since the particles are introduced periodically with time, there are many particles to trace. We assign a color to every cluster of external particles to keep track of the time when they were introduced in the *calculation*. The colors vary from magenta for those particles introduced early in the calculation, to blue for those introduced near the end of a detonation cycle. In Figure 3a, which corresponds to early times, only 12 clusters of external particles are visible. These clusters were introduced from $t = 0$ to $0.6 * 10^{-3}$ second, vary from magenta to red in color, and are tracking the expanding flow of the detonation products.

In Figure 3b the simulation results for the same case are shown for $t = 1.4 * 10^{-3}$ sec. The pressure contours show that a strong stagnation point develops on the axis of symmetry downstream at ≈ 25 cm as a result of a strong expansion of the Mach 0.8 external flow around the engine. At this time it is evident from the particle trajectories that most of the detonation products have left the chamber. Figure 3b also shows that only traces of the particles originating at the back wall of the detonation chamber are left in the computational domain. These particles advanced to the aft end of the chamber *and then following* the contraction of the over expanded detonation products, reversed their flow direction. The pressure contour plots in Figure 3b show the formation of an additional stagnation point at the closed end wall of the detonation chamber resulting from the inverse flow of the detonation products. The average pressure in the chamber is below ambient and is ≈ 0.55 atm.

The marker particles released outside and just above the nozzle exit show two distinct flow paths. One path

takes the flow past the stagnation point to the far right of the detonation chamber; this flow path is marked by the four upper particle traces. Another flow path, marked by three lower particles released close to the external wall of the chamber, are deflected from the stagnation region towards the detonation chamber exit. The magenta color of these particles indicates they were released at $t \approx 0.6 * 10^{-3}$ sec.

Figure 3c corresponds to the simulation time $t = 2.2 * 10^{-3}$ sec. The pressure inside the chamber has risen to ≈ 0.8 atm. The stagnation region at the closed end of the detonation chamber continues to develop and has produced a compression wave moving toward the open end of the chamber. The particles marking the movement of fresh air into the chamber show these to be well inside the chamber, with some reflecting from the end wall and contributing to the pressure at the second stagnation point. The circular motion of a few of the detonation products particles (red solid lines), indicates that the detonation products which did not expand with the first shock wave are now "trapped" inside the detonation chamber.

Figures 3d and 3e correspond to the end of the first cycle when the detonation chamber should be filled with fresh charge and ready for the next detonation. However, these figures indicate that the fresh air refill is not totally satisfactory for this chamber configuration. The marker particle paths indicate that the chamber refill is incomplete and at a time of $t = 4.7 * 10^{-3}$ sec the refill process has essentially stopped. As a result, only about a third of the detonation chamber volume has enough fresh air for the next detonation cycle.

In Figure 4 the total force and time averaged thrust generated by the device in the simulations just discussed, are shown as a function of time. The time averaged thrust is based on the total time for one cycle defined as $7.0 * 10^{-3}$ sec. This time is equivalent to a detonation frequency of 140 Hz. As seen in the figure, initially a very large force of $\approx 3.2 * 10^{3}$lb is felt on the end wall of the detonation chamber. This force is a result of the high pressure behind the detonation wave. It rapidly decays and at $t \approx 0.5 * 10^{-3}$ sec changes sign due to over expansion and dynamic pressure of the external flow. This effect is noticeable in the thrust data; the average thrust increases rapidly but decreases after reaching levels of ≈ 55 lbs. At the end of the simulation the thrust is actually negative ≈ -20 lbs. The average thrust for one cycle in this case will be ≈ 10 lbs.

The simulation just described has served to illustrate the information generated with the numerical simulations. For the remaining simulations, emphasis was placed on determining the effects of propagation direction of the main detonation wave, effects of inlet and valve geometry, detonation chamber geometry and Mach number. Many of the simulations produced unsatisfactory results from the point of view of ineffective fresh air refill and hence either not enough fresh charge for repetitive detonations or too slow a refill resulting in low detonation frequency. We give below examples of successful simulations at Mach 0.8, Case 2 and Mach 2, Case 3.

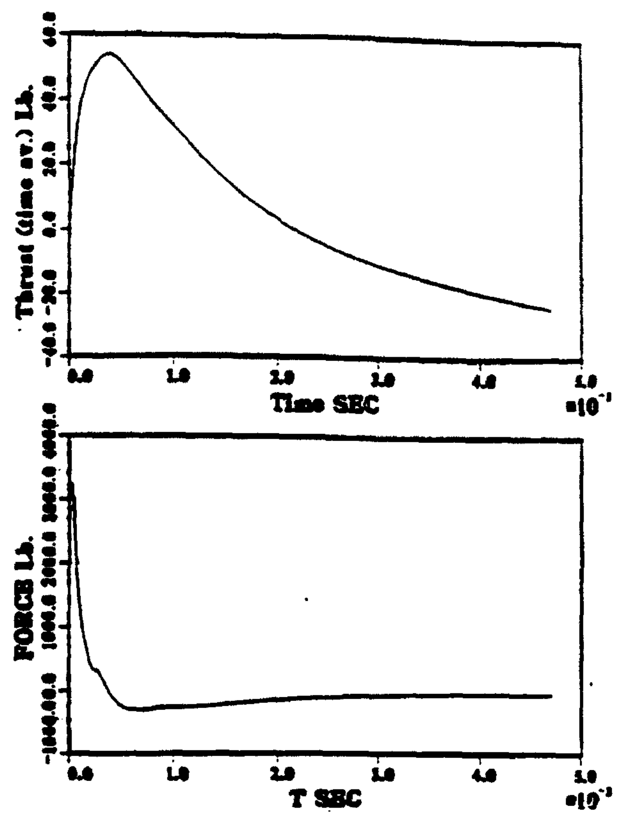


Figure 4. Time averaged thrust and force data from simulation of Case 1.

Case 2. The results from all simulations show that, irrespective of the inlet geometry, but with a straight nozzle and initial detonation position at the nozzle exit plane, sufficient thrust levels can be produced. A remaining problem in view of the objectives is to demonstrate that enough fresh air can be injected into the chamber to produce the required conditions for intermittent detonation at a frequency of 140 Hz. To accomplish this, we have considered a contoured inlet in the periphery of the end wall of the detonation chamber. The details of this inlet geometry and the computational grid are shown in Figure 5.

For the initial tests with this inlet no attempt was made to optimize the inlet geometry for a given flow regime. Figures 6a-f present results for the simulation of the chamber geometry shown in Figure 5.

The flight Mach number in this case is 0.8. The initial detonation wave is launched inwards and its energy parameters are the same as in all previous cases. In Figure 6a we see two distinct shock waves expanding into ambient air: one generated by expansion from the aft of the chamber and another produced by the expansion through the inlet. We also notice some particles tracing the motion of the detonation products flowing out through the inlet. In Figure 6b, at time $t = 0.7 * 10^{-3}$ sec., fresh air is noted entering the chamber through the inlet. At this time the dominant pressure in the chamber is 0.77 atm. Figure 6c shows that at the time $t = 1.4 * 10^{-3}$ sec, 3/4 of the detonation chamber is filled with fresh air. The strong air jet entering the chamber impinges the axis of symmetry, creating two large vortices which rotate in opposite directions. Such vortical motion would promote effective fuel-air mixing in the chamber. In Figure 6d, $t = 2.4 * 10^{-3}$ sec., the fresh air stream begins to exit the chamber. At this point the mixture inside the chamber has achieved the required conditions for the next detonation. This result translates to a sustained detonation frequency of $\approx$ 400 Hz. In Figures 6e-f we follow the later evolution of the flow pattern inside and outside the chamber. We observe strong air flow through the inlet with a strong recirculation pattern, which will assure fuel air mixing even if the fuel is injected into the chamber with a delay to sustain intermittent detonation at a lower frequency. In Figure 7 thrust and force simulated for the last case are shown as a function of time. First we notice that the maximum thrust for this case is $\approx$ 70 lbs., somewhat lower than for the cases with a very simple annular inlet and completely flat end walls.

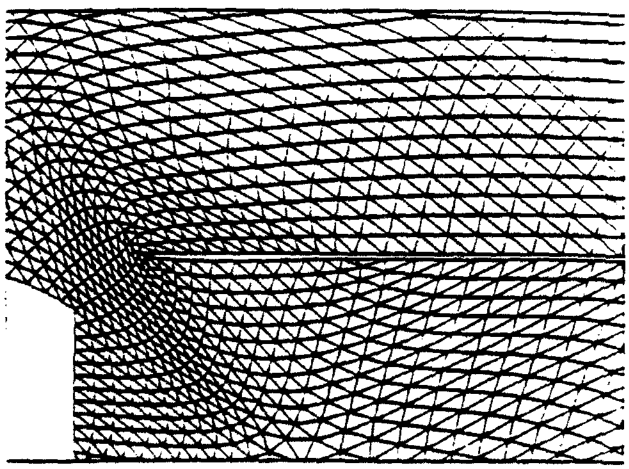


Figure 5. Computational grid for the inlet geometry used in simulation of Case 2.

This results from a reduction of the area normal to the propagation direction of the detonation wave due to the inlet geometry. It is surprising that the case with the inlet results in a reduction of the average thrust as a function of time that is almost the same as for a case without inlet at the same Mach number ($\approx$ 90 lb reduction without the inlet and 100 lb reduction with the inlet). This strongly indicates that the generic inlet we have just considered will not contribute significantly to the drag produced by the chamber dynamics and interaction with the ambient flow. The cycle average thrust generated by the PDE based on 150 Hz operation frequency in this case is $\approx$ 100 lbs. This value is somewhat larger than the thrust targeted for this study.

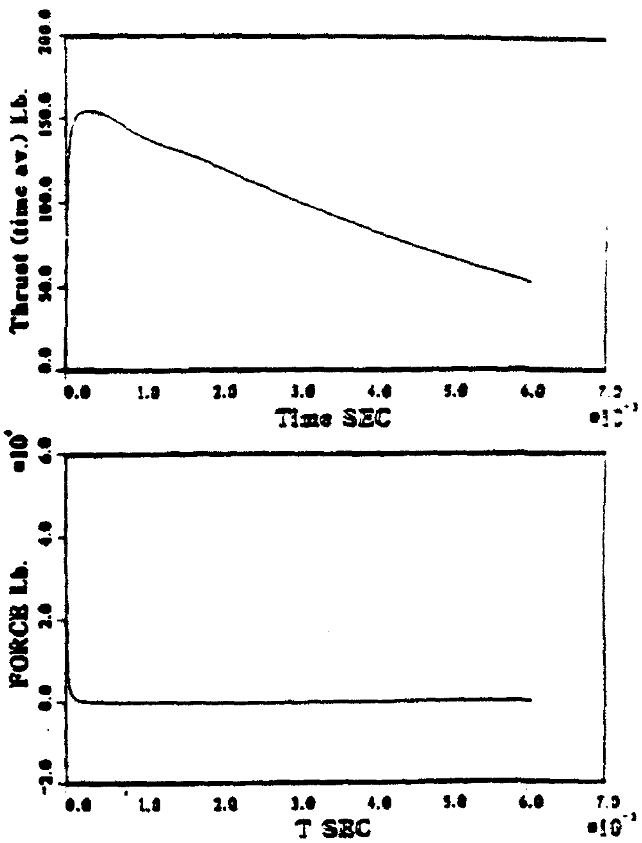


Figure 7. Time averaged thrust and force data from simulation of Case 2.

Case 3. Results for a Mach 2.0 simulation of the same geometry as in the previous simulation; but with a more geometrically complex inlet are shown in Figures 8a-b. The inlet geometry for this case was determined from near choked flow conditions in the throat region of the inlet. In addition to the pressure contours and particle paths, in this case we also show velocity vectors.

We observe in these figures that the detonation chamber is quickly filled with fresh air at the time $t = 1.3 * 10^{-3}$ sec., which corresponds to a detonation frequency of 700 Hz. In practice this high frequency will be difficult to realize, because of the mixing and initiation problems. In Figure 9 we show thrust and force results for this simulation. We observe that after $3.0 * 10^{-3}$ sec. the net average thrust is still 50 lbs.

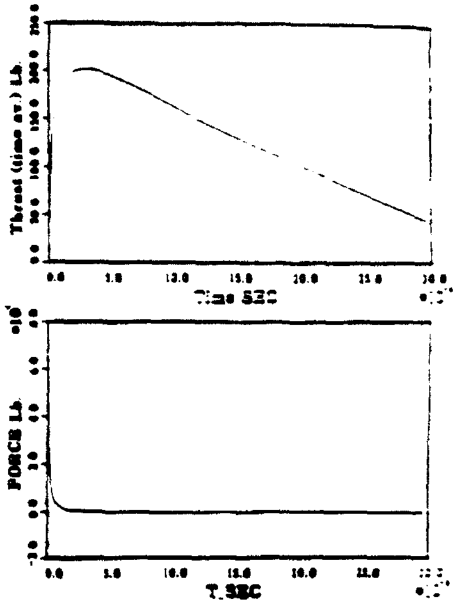


Figure 9. Time averaged thrust and force data from simulation of Case 3, 140 Hz detonation frequency.

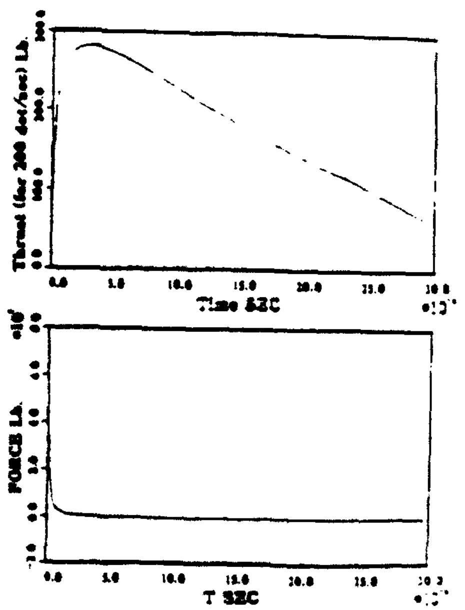


Figure 10. Time averaged thrust and force data from simulation of Case 3, 200 Hz detonation frequency.

The cycle averaged thrust based on 140 Hz detonation frequency, for this simulation is $\approx$ 70 lbs. However, as previously mentioned the fresh air refill time allows a much higher frequency of detonations. Figure 10 shows the same results as Figure 9, but calculated for a 200 Hz cycle frequency. In this case the maximum average thrust is $\approx$ 280 lbs. and the net cycle averaged thrust is $\approx$ 100 lbs. This result indicates the promising potential of the PDE concept for supersonic propulsion.

4. Conclusions

In this section we present our conclusions reached after carrying out a review of past research on detonative propulsion and a detailed numerical simulation of a generic pulsed detonation engine (PDE) device. The primary conclusion is that the PDE shows promising potential in providing primary propulsion for a range of present day aerodynamic vehicles such as target drones, PENAID missiles and other smart missiles that require loitering and throttling capability. The operating flight regimes of such a propulsion engine may extend from the low subsonic to supersonic regimes.

Most of the past attempts at producing thrust based on an intermittent detonation cycle were carried out with the same basic experimental set-up; namely, a long straight detonation tube employing forced fuel injection at the closed tube end. We have pointed out the reasons[25] why such a device cannot take proper advantage of the physical processes associated with detonative combustion. We have also indicated that, because of the conclusions reached during experiments with such devices, the development of intermittent detonative propulsion was adversely prejudiced and stalled at an early stage.

The experiments performed at the Naval Postgraduate School based on a self-aspirating mode of operation for a pulsed detonation thruster produced very useful results which, upon further examination, provide us with a route towards practical detonation engines of variable thrust levels that are both controllable and scalable. A generic PDE device based on the NPS experiments was conceptualized and served as the basic model for a comprehensive series of numerical simulations. The goal of the simulations was to understand the parametric dependence of the PDE device variables on propulsion performance such as thrust and detonation cycle frequency.

The principle conclusions drawn from the simulation results are as follows. First, the target thrust and cycle frequency of 60-90 pounds and 140 Hz, respectively, have been realized in the simulations. These target values were dictated by knowledge of present day requirements for planned aerodynamic vehicles such as PENAID devices. Before proceeding, it is appropriate to mention again that the performance of the PDE device is governed entirely by unsteady flow processes. None of the wave averaging effects which had been predicted

by previous studies were found and, it was shown dramatically that the internal (detonation, expansion, refill and mixing) flow processes are directly coupled to the external (shock formation, stagnation point formation, vortex shedding, etc.) flow processes. These two flows must be simultaneously analyzed if a reliable estimate of performance is to be determined. The present study is the first fully unsteady computational analysis of an intermittent detonation scheme with realistic geometry and external flow computed self-consistently.

The simulations further showed that the best thrust performance was realized when the full kinetic energy of the detonation wave was captured on the thrust surface (the closed end wall of the detonation chamber). This indicates that the detonation initiation must be controlled; the ignition must take place in the vicinity of the exit plane of the chamber resulting in initial propagation of the wave towards the chamber wall. The magnitude of the total and time averaged thrust is a strong function of the strength of the wave, the cross-sectional area of the end wall normal to the wave direction, and a weak function of the specific geometrical details of such variables as valve or inlet shape. The simulations also showed that for most situations involving simple inlets (flat cylindrically symmetric openings in the chamber external wall) the thrust data was independent of whether the valve intermittently opens or remains open during the full cycle. This leads to the possibility of a permanently open valve and a no moving parts manifestation of a PDE device. The thrust data indicates a dependence on the external flight conditions, e.g. Mach number. The Mach number plays a role in the wave drag that the geometry of the PDE will incur; the details of the valve and inlet configurations figure prominently in the total wave drag.

On the other hand the simulations showed that the timing of the fresh air refilling required to recharge the chamber for subsequent detonations is a strong function of the details of the valve and inlet geometry, the expansion of the combustion products, the resulting over-expansion of the chamber flow and, the external flow regime and interaction of the external flow with the internal flow. For subsonic flight, Mach 0.2-0.9, the fresh air entering the chamber comes from two separate principal flow processes; one comes from the flow through any valve or inlet and the other comes from the self-aspiration or reverse flow from the aft end of the chamber due to strong over-expansion. All these processes are interdependent, as reported in Section 3, and, in order to search for a given performance in a given device requires variation of many parameters. The simulation results obtained to date provide an understanding of the effects caused by variation of the above mentioned parameters, and with the information available we are able to conclude that a PDE propulsion unit can be optimized (although no optimization studies were carried out) for a given flight regime. In order to find an optimum configuration satisfying given performance over a wide flight regime a more extensive simulation study will be required. It was mentioned earlier that the simulations presented here were carried out under the assumption of inviscid flow; boundary layer effects were not included. The addition of boundary layers to the PDE engine inlets and valves, the only components where boundary layers will be significant, will lead to increased performance. Roughly the same amount of fresh air will flow into the over expanded detonation chamber but at a somewhat slower rate and in a pattern that will promote enhanced circulation and hence fuel/air mixing. We return to the issue of optimization below.

We give now results from sample performance calculations of the application of the PDE device to proposed aerodynamic vehicles such as a PENAID missile based on the results from our simulations. These predictions are based on point design data for an inlet geometry which has not been optimized. We believe that increased performance can be found through a systematic optimization of the PDE device characteristics. First we consider the Mach 0.8 case and the inlet described in Case 2.

The maximum operation frequency for the device is 400 Hz. The following performance is a consequence of the simulation data:

For a frequency of 100 Hz.:

Thrust	.79 lb.
Fuel flow rate	.0.025 lb./sec.
Fuel weight for 12 min	.18 lb.
Oxygen weight	.1.8 lb.
Fuel for detonation tube	.0.6 lb.
Total oxygen and fuel weight	.20.4 lb.
Total engine weight	.30.2 lb.
Specific fuel consumption	.1.14 lb./(lb.*hr.)

Assuming the PDE device geometry is kept fixed, a higher detonation frequency will result in a linear increase in thrust and fuel flow rate at the same specific fuel consumption. For example, if the detonation frequency is increased to 200 Hz., the performance data are:

Thrust	.157 lb.
Fuel flow rate	.0.05 lb./sec.
Fuel weight for 12 min	.36 lb.
Oxygen weight	.3.6 lb.
Fuel for detonation tube	.1.2 lb.
Total oxygen and fuel weight	.40.8 lb.
Total engine weight	.54.4 lb.
Specific fuel consumption	.1.14 lb./(lb.*hr.)

At lower Mach numbers, M=0.5, the maximum operating frequencies will be lower since the external dynamic pressure responsible for supplying fresh air to the chamber is also lower. For the device under considera-

tion here the maximum frequency is 250 Hz.

For a frequency of 100 Hz.:

Thrust	100 lb.
Fuel flow rate	0.025 lb./sec.
Fuel weight for 12 min	.18 lb.
Oxygen weight	.1.8 lb.
Fuel for detonation tube	.0.6 lb.
Total oxygen and fuel weight	.20.4 lb.
Total engine weight	30.2 lb.
Specific fuel consumption	.0.9 lb./(lb.*hr.)

Again, if the frequency is increased the thrust will increase linearly; operation at 200 Hz. yields:

Thrust	.200 lb.
Fuel flow rate	0.05 lb./sec.
Fuel weight for 12 min	.36 lb.
Oxygen weight	3.6 lb.
Fuel for detonation tube	1.2 lb.
Total oxygen and fuel weight	40.8 lb.
Total engine weight	.54.2 lb.
Specific fuel consumption	.0.9 lb./(lb.*hr.)

The examples of performance of PDE devices given above are based on point design conditions arising from the simulations discussed in Section 3 of this report. They cannot be extrapolated with any degree of reliability to other conditions or configurations. We conclude however, that the performance computed for the indicated device is encouraging from the point of view of thrust, thrust control, simplicity of the device (no moving parts) and specific fuel consumption (SFC). The specific fuel consumption computed above is competitive with present day small turbojet engines. The SFC for a PDE could be signficantly lower than for small turbojets (SFC's for small turbojets are in the range of 1.8-2.0 lb./(lb.*hr.)). Thus, for a given mission and vehicle, a PDE propulsion unit would be more fuel efficient resulting in increased range. Moreover, if the expected thrust control in PDE's is realizable, it may be possible to produce propulsion units that can slow down, loiter and maneuver and finally accelerate to full thrust again rapidly.

A final conclusion can be made concerning the application of PDE's to supersonic vehicles. As shown in the simulations the ability to refill the detonation chamber with fresh air charge is a very strong function of valve and inlet geometry. Refilling may also be somewhat enhanced by the self-aspiration effect, but; to a much less extent than in the subsonic case. The example of supersonic operation discussed in Section 3 shows that care must be taken in design of the inlet or valve configuration. The flow in the chamber must allow for refill and fuel/air mixing. More than likely choked flow conditions will be required at the inlet entrance to the chamber. This could lead to complications in the design of a PDE with simple geometry; choked flow conditions are a function of the external Mach number and a fixed inlet will be optimal only for a small range of the operating envelope. On the other hand, if a given vehicle is to fly at supersonic speeds and is launched at supersonic speeds, this problem may not appear. Further, if the given vehicle is launched at subsonic speeds and a booster is used to bring it up to the required supersonic operating speed, the problem may again not appear. We conclude that the PDE has potential for the supersonic flight regime and it is not excluded that a configuration can be found which will operate over the flight regimes $0.2 <$ Mach number < 3 in a fuel efficient manner.

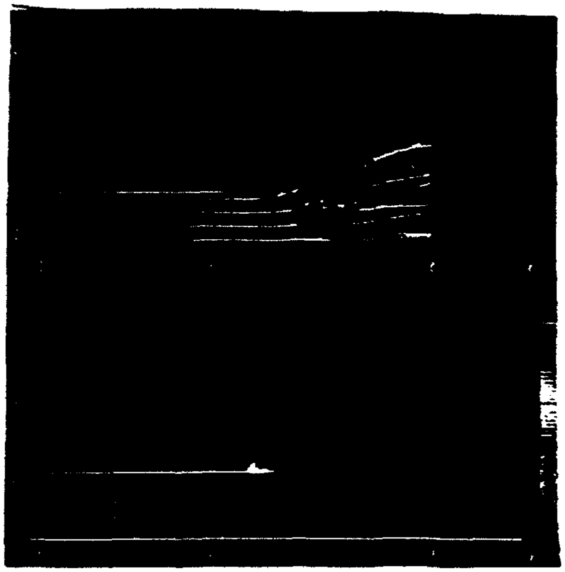
Finally it is appropriate to speculate that the PDE concept is a candidiate for a hybrid propulsion device. Consider the following scenario. At low altitudes, up to 30-50 km, and at speeds ranging from low supersonic to hypersonic ($2 <$ Mach number < 10) an air breathing engine can operate. Above these conditions air breathing is not effective and rocket propulsion is required. A PDE can operate in an air breathing mode as long as the external conditions allow it, and when no longer possible, the detonation chamber may be considered a rocket chamber in which detonation occurs with the fuel and oxygen supplied from on-board storage. Similar considerations have been made for NASP propulsion; serious penalties are made in that large quantities of fuel must be carried. However, for vehicles such as the current Pegasus, a PDE propulsion device may be attractive from the point of view of thrust control over a large portion of the flight envelope.

Acknowledgements

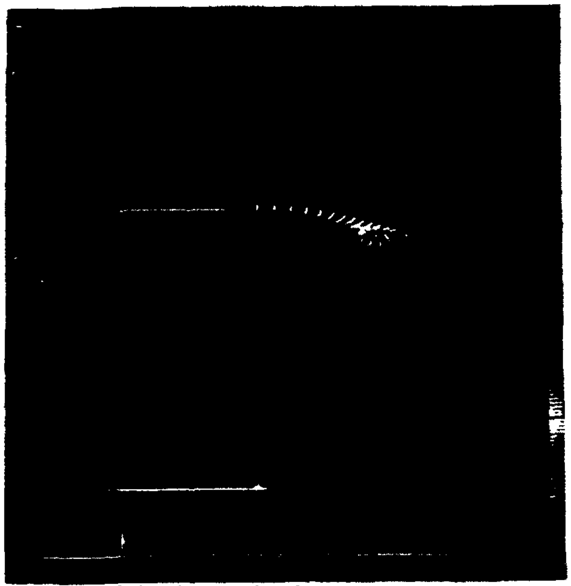
The authors would like to express appreciation to Drs. Adam Drobot and Aharon Friedman of SAIC for helpful suggestions and advice during the course of this work. The work reported on here was supported by DARPA under contract N66001-88-D-0088. The authors would like to thank Col. Doc Daugherty for his encouragement and interest in this project.

References

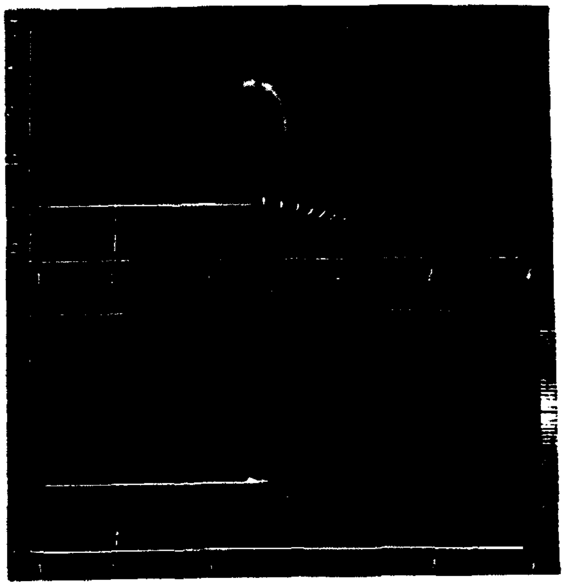
1. Stodola, A., Steam and Gas Turbines. McGraw-Hill Inc., 1927.
2. Zipkin. M. A., and Lewis G. W., "Analytical and Experimental Performance of an Explosion-Cycle Combustion Chamber of a Jet Propulsion Engine." NACA TN-1702, Sept. 1948.
3. Shultz-Grunow, F., "*Gas-Dynamic Investigation of* the Pulse-Jet Tube," NACA TM-1131, Feb. 1947.
4. Zinn, B. T., Miller, N. Carvelho, J. A., and Daniel B. R.,"Pulsating Combustion of Coal in Rijke Type Combustor," 19th International Symposium on Combustion, 1197-1203, 1982.
5. Hoffmann, N., "Reaction Propulsion by Intermittent Detonative Combustion," Ministry of Supply, Volkenrode Translation, 1940.
6. Nicholls, J. A., Wilkinson, H. R. and Morrison, R. B. "Intermittent Detonation as a Thrust-Producing Mechanism," Jet Propulsion, **27**, 534-541, 1957.
7. Dunlap, R., Brehm, R. L. and Nicholls, J. A., " A Preliminary Study of the Application of Steady State Detonative Combustion of a Reaction Engine", ARS J., **28**, 451-456, 1958.
8. Nicholls, J. A., Gullen, R. E. and Ragland K. W., "*Feasibility Studies of a Rotating Detonation Wave* Rocket Motor," Journal of Spacecrafts and Rockets, **3**, 893-898, 1966.
9. Adamson, T. C. and Olsson, G. R., "Performance Analysis of a Rotating Detonation Wave Rocket Engine," Astronautica Acta, **13**, 405-415, 1967.
10. Shen. P. I., and Adamson, T. C., "Theoretical Analysis of a Rotating Two-Phase Detonation in Liquid Rocket Motors," Astronautica Acta, **17**, 715-728, 1972.
11. Krzycki, L. J., Performance Characteristics of an Intermittent Detonation Device, Navweps Report 7655, U. S. Naval Ordnance Test Station, China Lake, California 1962.
12. Matsui, H., and Lee, J. H., " On the Measure of the Relative Detonation Hazards of Gaseous Fuel-Oxygen and Air Mixtures," Seventeenth Symposium (International) on *Combustion*, 1269-1280, 1978.
13. Korovin, L. N., Losev A., S. G. Ruban and Smekhov, G. D. "Combustion of Natural Gas in a Commercial Detonation Reactor," Fiz. Gor. Vzryva **17**, 86 (1981).
14. Smirnov, N. N., Boichenko, A. P., " Transition from Deflagration to Detonation in Gasoline-Air Mixtures," Fiz. Gor. Vzryva **22**, 65 (1986).
15. Lobanov, D. P., Fonbershtein, E. G., Ekomasov, S. P., "Detonation of Gasoline-Air Mixtures in Small Diameter Tubes," Fiz. Gor. Vzryva **12**, 446 (1976).
16. Back, L. H., "Application of Blast Wave Theory to Explosive Propulsion," Acta Astronautica 2, 391 (1975).
17. Varsi, G., Back, L. H., and Kim, K., "Blast Wave in a Nozzle for Propulsion Applications," Acta Astronautica, 3, 141 (1976).
18. Kim, K., Varsi, G., Back and L. H., "Blast Wave Analysis for Detonation Propulsion," AIAA Journal **10**, Oct. 1977.
19. Back L. H., Dowler, W. L. and Varsi, G., "Detonation Propulsion Experiments and Theory," AIAA Journal **21**, Oct. 1983.
20. Eidelman, S., Shreeve, R. P., "Numerical Modeling of the Nonsteady Thrust Produced by Intermittent Pressure Rise in a Diverging Channel," ASME FED Multi-Dimensional Fluid Transient **18**, 77 (1984).
21. Eidelman, S., "Rotary Detonation Engine," U.S. Patent 4 741 154, 1988.
22. Helman, D., Shreeve, R. P., and Eidelman, S., "Detonation Pulse Engine,", AIAA-86-1683, 24th Joint Propulsion Conference, Huntsville, 1986.
23. Monks, S. A., "Preliminary Assessment of a Rotary Detonation Engine Concept," MSc Thesis, Naval Postgraduate School, Monterey, California, Sept. 1983.
24. Camblier, T. L. and Adelman, N. G., "Preliminary Numerical Simulations of a Pulsed Detonation Wave Engine," AIAA-88-2960, AIAA 2?th Joint Propulsion Conference, Boston 1988.
25. Eidelman, S., W. Grossmann, I. Lottati, "A Review of Propulsion Applications of the Pulsed Detonation Engine Concept," AIAA 89-2466, AIAA/ASME/SAE/ASEE 25th Joint Propulsion Conference, Monterey, CA, July 10-12, 1989 (to be published in AIAA Journal of Propulsion).
26. Lottati, I., S. Eidelman, A. Drobot, "A Fast Unstructured Grid Second Order Godunov Solver (FUGGS)," Paper AIAA 90-0699, 28th Aerospace Sciences Meeting, Reno, CA, Jan 8-11, 1990.



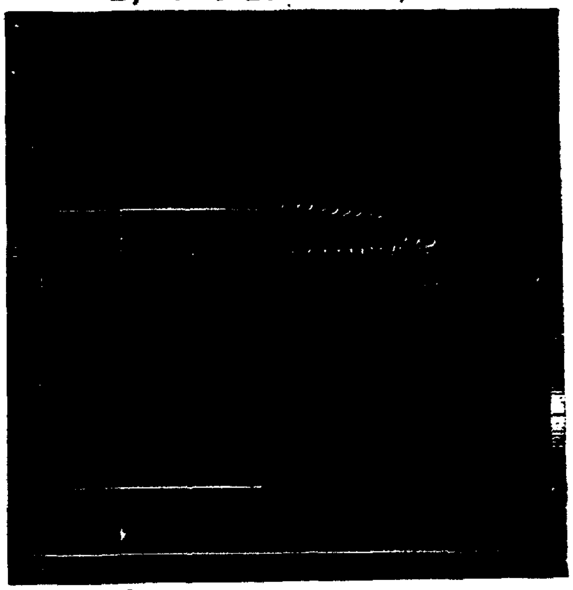
a) t = 0.64 msec,



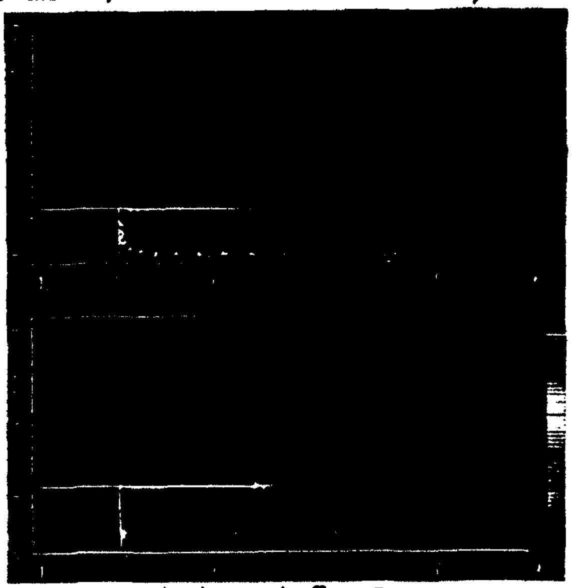
b) t = 1.4 msec,



c) t = 2.2 msec,



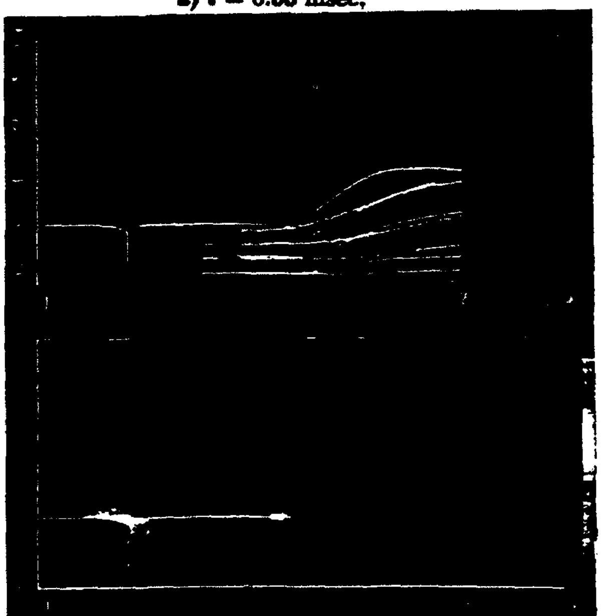
d) t = 3.4 msec,

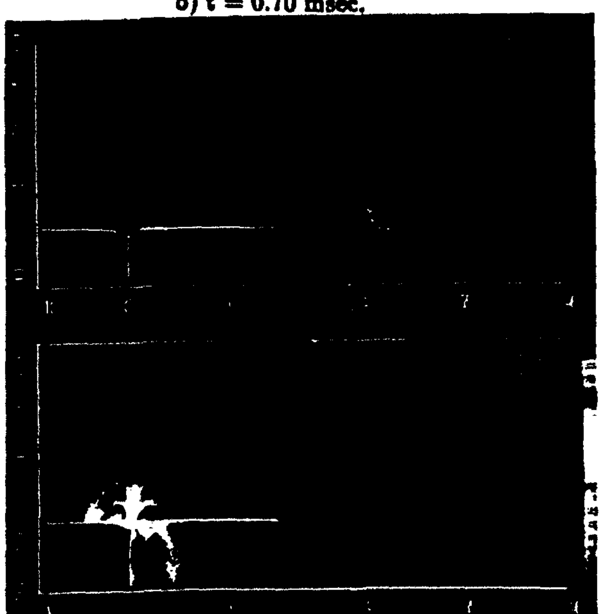


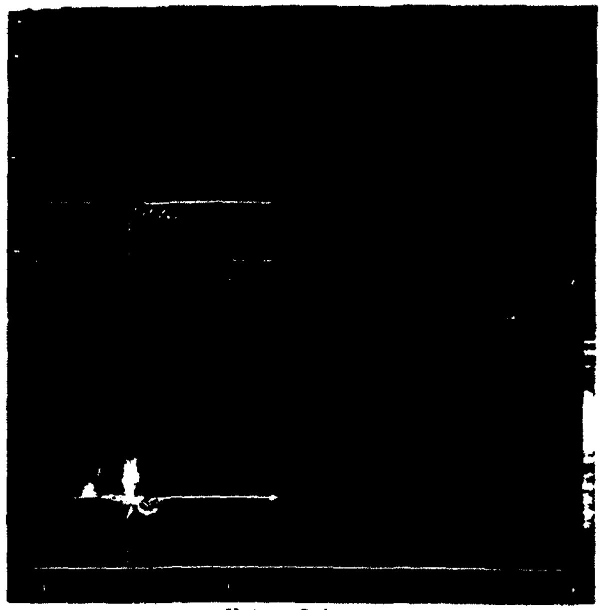
e) t = 4.7 msec,

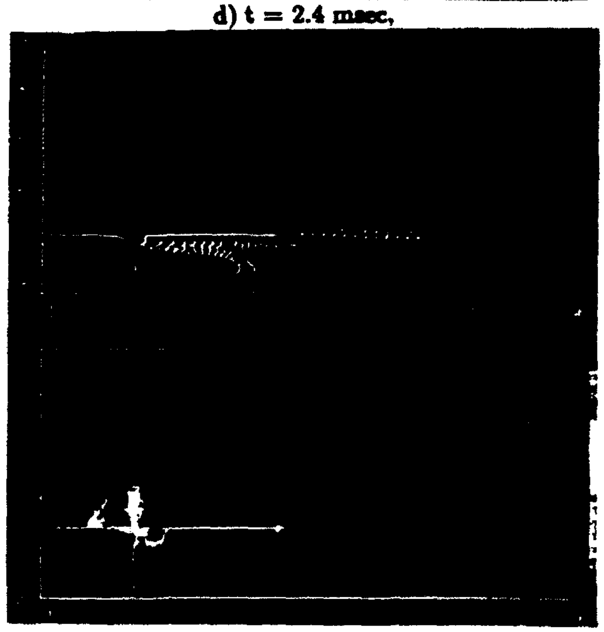
Figure 3 Pressure contours and marker particle paths for Case 1, M = 0.8, no inlet, inward initial detonation location.


a) t = 0.33 msec,


b) t = 0.70 msec,


c) t = 1.4 msec,


d) t = 2.4 msec,


e) t = 2.7 msec

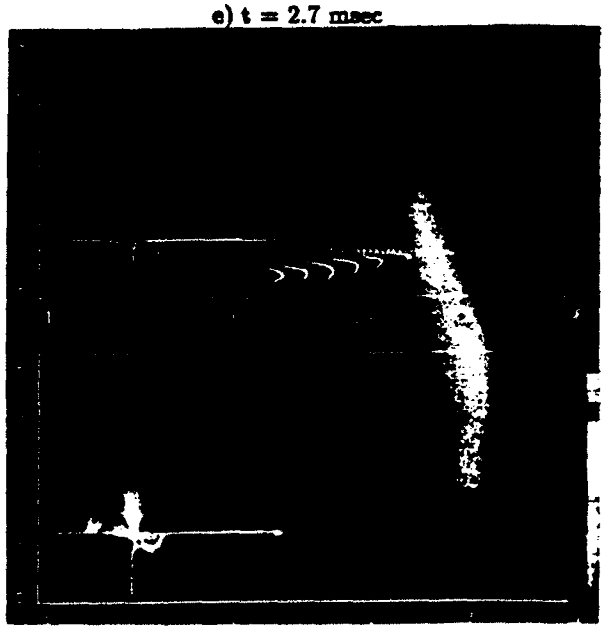
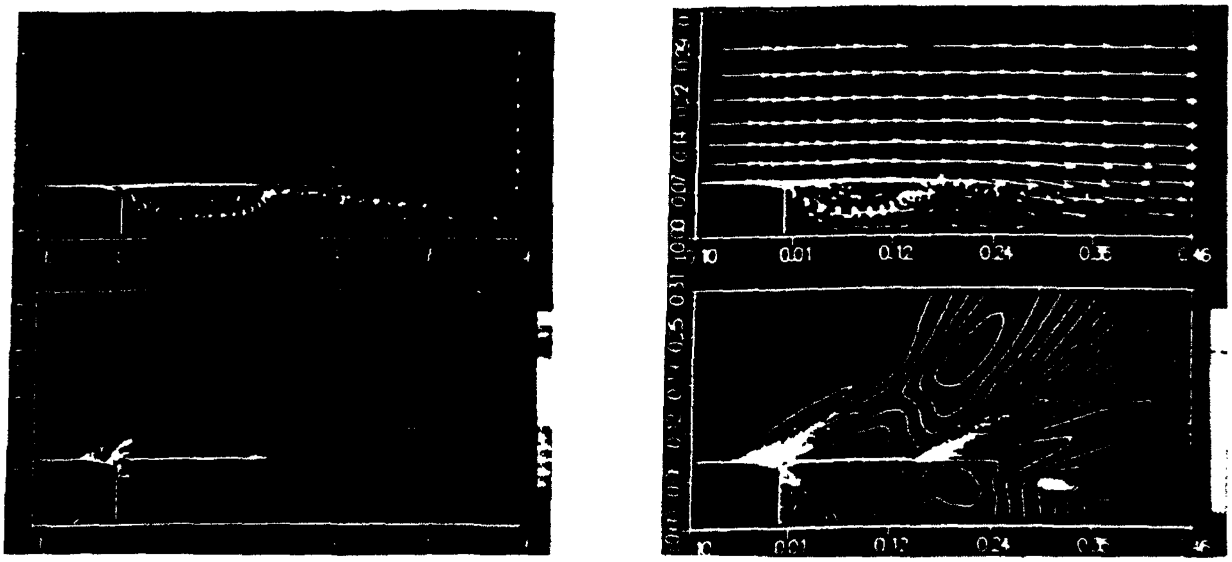

f) t = 4.1 msec

Figure 6 Pressure contours and marker particle paths for Case 2, M = 0.8, inlet geometry of Figure 5, outward initial detonation location.



a) t = 1.3 msec,　　　b) t = 2.1 msec,

Figure 8 Pressure contours and marker particle paths for Case 3, M = 2.0, sculptured inlet, outward initial detonation location.

AIAA

AIAA-90-0699

A FAST UNSTRUCTURED GRID SECOND ORDER GODUNOV SOLVER (FUGGS)

Itzhak Lottati, Shmuel Eidelman, and Adam Drobot
Science Applications International Corporation
McLean, VA

28th Aerospace Sciences Meeting

January 8-11, 1990/Reno, Nevada

A FAST UNSTRUCTURED GRID SECOND ORDER GODUNOV SOLVER (FUGGS)

Itzhak Lottati, Shmuel Eidelman, and Adam Drobot

Applied Physics Operation
Science Applications International Corporation
McLean, Virginia 22102

ABSTRACT

We describe a new technique for solving Euler's equations on an unstructured triangular grid with arbitrary connectivity. The formulation is based on Godunov methods and is second order accurate. The use of a unique data structure leads to an easily vectorized and parallelized code with speed and memory requirements comparable to those found with logically structured grids. The new algorithm has been tested for a wide range of flow conditions ranging from low speed subsonic flow to hypersonic flow with Mach number 32. The results obtained are comparable to or better than those obtained with leading flow solvers in all of the regimes tested. The code contains no free parameters and can be used in complex flow problems where a variety of flow conditions may be encountered.

INTRODUCTION

This paper introduces a new second order algorithm for solving the Euler equations on an unstructured grid, using an approach based on first and second order Godunov methods. The formulation presented here leads to an extremely efficient and fast Flow Solver which is fully vectorized and easily lends itself to parallelization. The low memory requirements and speed of the method are due to the use of a unique data structure.

Explicit hydrodynamic numerical algorithms are easily adapted to Massively Parallel Computers (MPC) for logically structured grids. This is a consequence of the fact that the calculation of the flow quantities are locally determined. For logically structured quadrilateral grids, the integration algorithm or Flow Solver computes the new flow values at the grid cell nodes (or centers) using the values of the flow parameters from the previous timestep employing four or more of the adjacent nodes. Higher order structured solvers are usually more computationally intensive, but retain the ability of the solver algorithm to be separated into several distinct steps, each of which can easily be vectorized and parallelized.

Until recently, most CFD simulations were carried out with logically structured grids and consequently vectorization and/or parallelization did not present a problem. The increased need for simulation of flow phenomena in the vicinity of complex geometrical bodies and surfaces has led to the emergence of CFD codes based on logically unstructured grids. The most successful of these unstructured grid codes are based on finite elements [1-6] or finite volume [7-12] methods.

Unstructured grid CFD computations in two-dimensions usually decompose the simulation domain into triangular elements. The physical location of the triangular elements and the accompanying list of vertices and edges is random with respect to the element index, making it necessary to maintain an indirect addressing system containing the connectivity information.

Calculations performed on unstructured grids evolve around the elemental grid shape (e.g. the triangle for two-dimensional problems); there is no obvious pattern to the order in which the local integrations should be performed. Explicit integration of hydrodynamic problems on an unstructured grid requires that a logical substructure be created identifing the locations in the global arrays of all the local quantities necessary for the integration of one element. As a result, there is usually a significant cost in computational efficiency, memory requirement, and code complexity. Approaches to vectorization for the conventional unstructured grid methods have concentrated on rearrangement of the data structure in a manner such that these locally centered data structures appear as global arrays. This can be done to some extent using machine dependent Gather-Scatter operations. Additional optimization can be achieved using localization and search algorithms [13]. However, these methods are complex and result in marginal performance. To date, most optimized unstructured codes have run considerably slower and require an order of magnitude more memory per grid cell than their structured counterparts. Parallelization of the conventional unstructured codes is even more difficult, and there is very little experience with unstructured codes on Massively Parallel Computers.

The method we describe in this paper overcomes these difficulties and results in code with speed and memory requirements comparable to those found in structured grid codes. Moreover, the ability to construct grids with arbitrary resolution leads to a flexibility in dealing with complex geometries which is not attainable with structured grids. The essence of the method is based on independent flux calculation across the edges of a dual baricentric grid, followed by node integration. This approach allows the flux and integration calculations to be performed on global arrays, coded as large vector loops, and is independent of element position on the unstructured grid.

In this paper we discuss our choice for data structure, the numerical algorithm (for first and second order solvers), and the results of test calculations. In realistic CFD

problems the physical domain may contain regions that span all flow regimes. It is very important that the numerical code be able to perform well over the full range of flow parameters with no a priori code "refinement." This is especially true of complex problems where flow conditions cannot be easily assessed in all subdomains. A robust code has clear advantages if it is possible to apply it with confidence under such circumstances. We have chosen four test problems to benchmark and validate the FUGGS code. These include: i) a subsonic flow case for steady flow with M = 0.2; ii) supersonic steady flow with M = 2.0; iii) hypersonic steady flow with M = 32.0; and iv) transit supersonic flow in a shock tube and over a wedge. For all of the test cases the method developed resulted in accurate solutions comparable to or better than reported in the literature by other leading CFD researchers. At the same time, the combination of using unstructured methods and our specific implementation yielded the lowest utilization of computer time and memory needed to achieve a given level of accuracy.

DATA STRUCTURE

On an unstructured grid, the data that describes the connectivity of a grid and the associated geometrical coefficients can represent a considerable overhead on memory usage. We have implemented a rather simple data structure which permits efficient finite difference integration of fluid quantities with only one level of indirection. For two dimensions, the data consists of lists of vertices, edges, and triangles. The physical quantities are stored at vertex locations. The vertex list consists of: the vertex positions (x,y), the fluid variables, the vertex volume, and workspace. The edge data is composed of: the addresses of the two vertices which form an edge, a vector which indicates the normal to the face that crosses an edge, the face area, and storage for the fluxes. The face is formed by joining the baricenters of the adjoining triangles which lie along the edge. This is the only data required in performing an iteration step. For convenience and ease of diagnostics, we have also maintained a list of triangles, including the positions of the baricenters, and the addresses of both the vertices and edges which form a triangle.

The data structure is compatible with algorithms which decompose the solution of the Euler equations into two steps. The first is determination of the fluxes. This can be realized by a loop over edges where the fluid quantities along the edge can be fetched through the indirect addressing of vertex data. The second step is to integrate the fluxes which contribute to the vertex. There are two options here: one is to maintain a list of flux elements at each vertex and to perform a loop over vertices and then fluxes to each vertex; the other is to again have a loop over edges where each contribution to the vertex is done as a random fetch and store using the appropriate vertex addresses stored by each edge.

BASIC INTEGRATION ALGORITHM

We begin by describing the first order Godunov method for a system of two-dimensional Euler equations written in conservation law form as

$$\frac{\partial \vec{Q}}{\partial t} + \frac{\partial \vec{F}}{\partial x} + \frac{\partial \vec{G}}{\partial y} = 0 , \qquad (1)$$

where,

$$\vec{Q} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ e \end{pmatrix}, \quad \vec{F} = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ (e + p)u \end{pmatrix}, \quad \vec{G} = \begin{pmatrix} \rho \\ \rho v u \\ \rho v^2 + p \\ (e + p)v \end{pmatrix} .$$

Here u and v are the x and y velocity vector components, p is the pressure, ρ is the density and e is the total energy of the fluid per unit volume. It is assumed that a mixed (initial conditions, boundary conditions) problem is properly posed for the set of equations (1), and that an initial distribution of the fluid parameters is given at t = 0, and the boundary conditions defining a unique solution are specified for the computational domain.

We seek a solution of the system of equations represented by Eq. (1) on a computational domain which is decomposed into triangles with arbitrary connectivity. An overwhelming advantage of using this method of domain decomposition is the ability to resolve extremely complicated geometries where the characteristic dimensions of subdomain features can vary over many orders of magnitude.

As an example, Figure 1 shows an unstructured triangular grid used in the simulation of flow for a new generation of the wide body tennis rackets with 21 cross string rows represented as solid circles and a tennis ball. In Figure 1a a blowup of the region near the racket surface is shown. This example illustrates the ability of the unstructured grids to represent geometry of arbitrary complexity and with localized resolution.

There are several options possible for storing physical information on an unstructured triangular grid: i) vertex centered; and ii) triangle centered on either a baricentric or Voronoi node. The selection of a specific grid structure offers two contrasting approaches. The first is to place the effort on creating an optimal grid, as is the case with Voronoi - Delauney meshes, while the second is to rely on the robustness of the integration algorithm. For complex configurations it is more difficult to achieve an optimum Voronoi-Delauney mesh and we have therefore opted for a simple baricentric grid.

This grid can always be constructed for a set of arbitrary triangles. The integration algorithm we have constructed can easily be implemented for both vertex and baricentered control volumes. Figure 2 displays a fragment of such a computational domain with the corresponding dual grid. The secondary or dual grid is formed by connecting the baricenters of the primary mesh, thus forming finite polygons around the primary vertices. Independent of the remarks made in Ref. 17 concerning usefulness of Dirichlet tessellation, we have confirmed that the best practical representation of the integration volume is obtained when the dual grid is formed by connecting baricenters of the triangles.

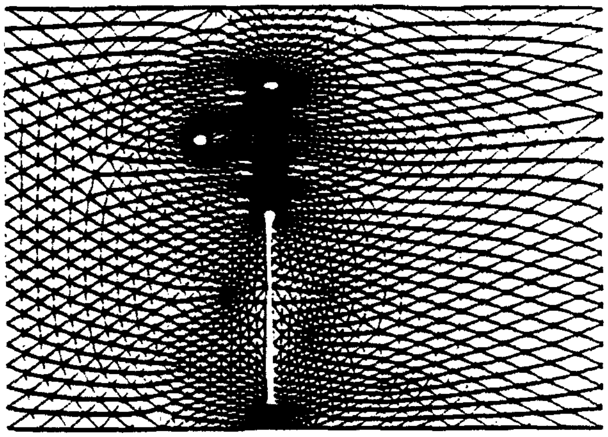


Figure 1 An illustration of the ability of a triangular grid to efficiently resolve the geometric complexity and features of objects with disparte spatial scales.

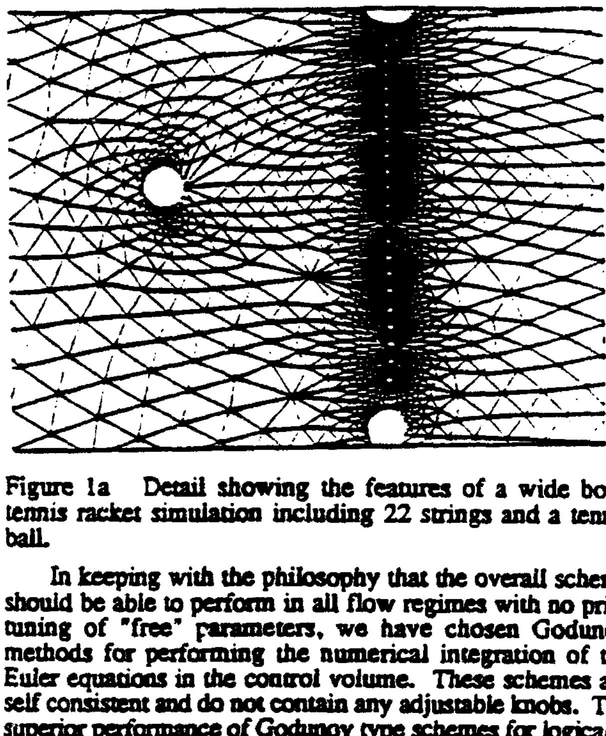


Figure 1a Detail showing the features of a wide body tennis racket simulation including 22 strings and a tennis ball.

In keeping with the philosophy that the overall scheme should be able to perform in all flow regimes with no prior tuning of "free" parameters, we have chosen Godunov methods for performing the numerical integration of the Euler equations in the control volume. These schemes are self consistent and do not contain any adjustable knobs. The superior performance of Godunov type schemes for logically structured grids is well documented and the advantages can be readily realized on triangular grids. Integration by the Godunov method consists of two basic steps: i) determination of the fluxes on the faces of the dual grid, which defines the control volume. This is accomplished by solving a set of one-dimensional Riemann problems along triangle edges; and ii) integration of the system of partial differential equations, which now amounts to a summation of all the fluxes for the vertex-centered control volume at each timestep.

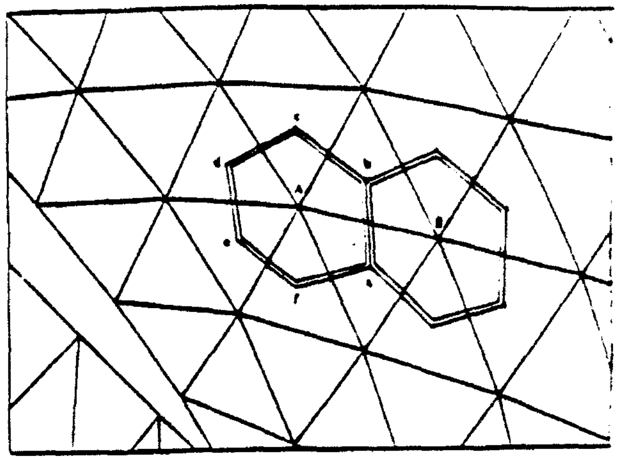


Figure 2 A triangular grid and the baricentered dual grid which defines the control volume. The fluxes are found on the faces of the control volume on each edge joining adjacent vertex points.

To define the fluxes flowing into the control volume shown in Figure 2, it is necessary to solve the Riemann problem along every edge of the primary grid and transverse to the faces of the dual grid. For example, to define the flux through the face **ab**, we solve the Riemann problem between vertices A and B. The solution of this problem is in coordinates local to the face of the dual grid **ab** so that the tangential component of velocity will be directed along **ab**. Implementation of our approach requires maintaining strict consistency when defining the "left" and "right" states for the Riemann problem at the faces **ab**, **bc**, **cd**, **de**, **ef**, and **fa**. For this reason we define not only the location of the vertices and areas of the faces but also the direction of the areas with respect to the primary grid edges. For the clockwise integration pattern in a polygon, vertex A will be the "left" state for all the Riemann problems related to this point and the neighbors will represent the "right" sides of the diaphragm.

It is easy to see that the flux calculation is based on information at only two nodes and requires simple geometrical parameters defining the face of the secondary grid which dissects the line connecting the two points. Thus, we can find all the values needed for the flux calculation in one vector loop over all edges of the primary grid without requiring any details related to the geometrical structures which these edges form. This in turn assures parallelization or vectorization of the algorithm for the bulk of the calculations involving the Riemann solver, which provides the first order fluxes.

The only procedure not obviously parallelizable is the integration of the fluxes for the flow variables at the vertices of the grid.

This operation requires a random fetch and store which can lead to conflicts that impair both parallelization and vectorization. Several common methods have been developed to deal with this difficulty. A practical approach is to split the integration of the fluxes into a small number of independent loops through the use of "edge" coloring. The number of loops necessary is determined by the maximum connectivity of any vertex in the domain and is

usually 7 or 8. Each of these loops is usually large enough not to impair vectorization. At this stage all the fluxes are added with their correct sign corresponding to the chosen direction of integration within the cell. The amount of computation required here is minimal since the fluxes are known and need only to be multiplied at each time step by a simple factor and added to the vertex quantity. This simple procedure results in a first order solver which is fully vectorized.

SECOND ORDER INEGRATION ALGORITHM

The second order solver is constructed along lines similar to that of the first-order method. At each cell face the Riemann problem is solved for the appropriate pair of left and right conditions. The solution to the Riemann problem is then used in calculation of fluxes, which are to be integrated later to advance to the next integration step. The extension to second order is achieved by using extrapolation in space and time to obtain time-centered left and right limiting values as inputs for the Riemann problem. The basic implementation of the method for finding the second order accurate fluxes is the same as for the one dimensional case and can be found in Refs. 14 and 16. The difference is the method of obtaining linear extrapolations for the flow variables as a first guess of their value on the faces of the dual grid. To obtain the initial guess we need to know the gradient of each gasdynamical parameter U at the vertices of the primary mesh. The value ∇ U can be evaluated by using the linear path integral around the finite volume associated with the vertex. For vertex A in Figure 2:

$$\int_A \vec{\nabla} U dA = \oint_l U \hat{n} dl$$

where integration along the path l in this case is equivalent to integration along the lines **ab**, **bc**, **cd**, **de**, **ef**, **fa**, and where n is a unit vector pointing outward from the control volume centered at A and normal to the integration path l. Knowing the gradient of the gasdynamic parameter in the volume related to vertex A allows us to extrapolate the values of this parameter at any location within the volume. This permits us to evaluate the first guess for U at the edges of the dual grid. The rest of the implementation of the second order algorithm is the same as described in Refs. 8 and 9. This includes monotonicity constraints similar to those introduced by VanLeer [15] and characteristic constraints described in Refs. 14 and 16.

A schematic of the basic steps for the second order algorithm is shown in Figure 3. This consists of five steps. They are: i) the calculation of the linearly extrapolated values at each side of the control volume faces using the left and right adjacent vertices and the values for each quantity and its gradient; ii) limiting the quantities obtained based on a monotonicity constraint; iii) a further limiter based on the solution of a one dimensional characteristic equation, which assures that the extrapolation does not violate the characteristics; iv) solution of the Riemann problem for the final extrapolated values with the limiters applied; and v) integration over the control volume.

The advantages of the method described will be demonstrated in the following section. The inclusion of the characteristic limiter has significantly improved the treatment of contact discontinuities and is the first such implementation on a triangular mesh.

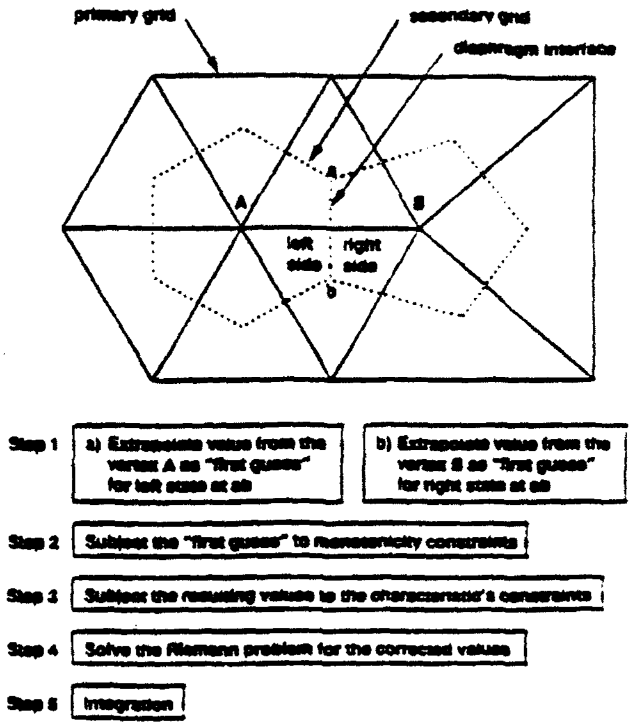


Figure 3 Schematic for stepwise implementation of the second order Godunov method on an unstructured grid.

RESULTS FOR TEST PROBLEMS

We have picked a set of test problems to demonstrate the performance of the FUGGS code for unsteady shock wave problems, and for subsonic, supersonic and hypersonic steady state flows. The cases in the chosen examples have analytical solutions that can be used to quantify the accuracy of the code and to validate the performance. This set of problems is frequently used by other CFD researchers and forms a basis for comparing FUGGS with other techniques.

a. Unsteady Shock Problem

As a first test we have chosen a case of planar shock wave propagation in a channel.

A section of the grid used for this test problem is shown in Figure 4. The total grid contained ~ 2000 vertices with a resolution of 100 points in the direction of propagation. We simulated a simple shock tube problem on

this grid where the gasdynamic parameters to the left and right of the diaphragm have the following values:

$$P_l = 1.0;\ \rho_l = 1.0;\ U_l = 0;$$
$$V_l = 0;\ \gamma_l = 1.4;$$
$$P_r = 0.1;\ \rho_r = 0.125;\ U_r = 0;$$
$$V_r = 0;\ \gamma_r = 1.4.$$

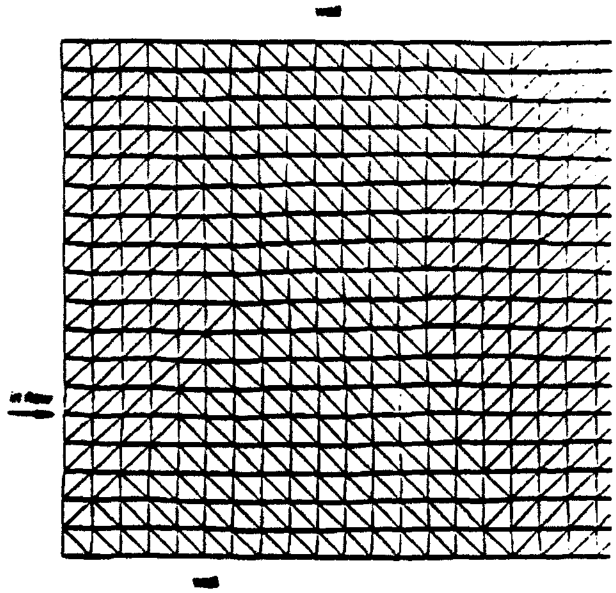


Figure 4 Section of grid from the unsteady shock problem.

This one dimensional problem was simulated on a rather ill formed grid (from the viewpoint of connectivity). Consequently the quality of the solution depended on the flow solver for accuracy. For the triangular shape of the elementary cell, planar shock and rarefaction waves generated by the solution always propagate at conflicting angles with respect to four out of the six edges of the control volume. The triangular grid chosen for this simple test problem therefore indicates the accuracy of FUGGS for shock waves of arbitrary shape and orientation moving through the computational domain. The density distribution found from three different versions of FUGGS is shown in Figure 5 as a function of x along the median cross section of the grid. The three cases are: i) first order Godunov method; ii) second order Godunov; and iii) second order Godunov with the characteristic limiter. The data displayed in the figure represents a loss of resolution due to interpolation of the actual grid values to the projected midsectional line. It is clear from Figure 5 that the final implementation of FUGGS with characteristic constraints yields the best results for contact discontinuities. The code also maintains the one dimensional structure for the shock in all three cases described above. The accurate representation of the density is also typical for all the other gasdynamic parameters.

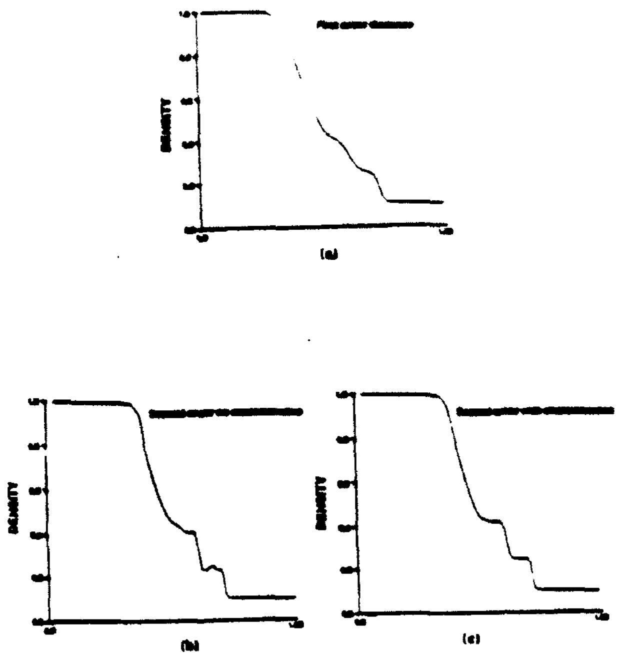


Figure 5 Solution to the density distribution of shock problem with three different versions of FUGGS: a) First Order Godunov, b) Second Order Godunov without characteristics and c) Second Order Godunov with characteristics.

b. Shock on Wedge

Here we demonstrate the performance of the methods for steady supersonic flow simulations. An analytical solution from oblique shock wave theory exists and can serve as an unambiguous comparison with the numerical simulation.

The initial grid for the shock on wedge problem is shown in Figure 6. This gridding results in ~ 500 vertices and ~ 800 triangles. The wedge angle in Figure 6 is 10°. The incoming flow enters the computational domain normal to the left boundary at Mach number M=2. Figures 7a and 7b show isomach lines for the steady flow solution from the first and second order Godunov solvers on the original grid. Comparing these two solutions we can see that the second order solution substantially improves the shock resolution. However, it is obvious that the grid density is too small to adequately resolve the oblique shock wave in both cases.

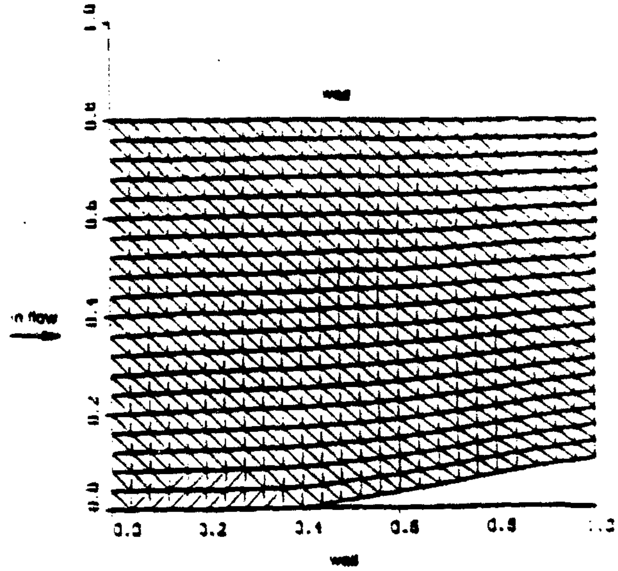


Figure 6 Coarse grid for shock on wedge problem.

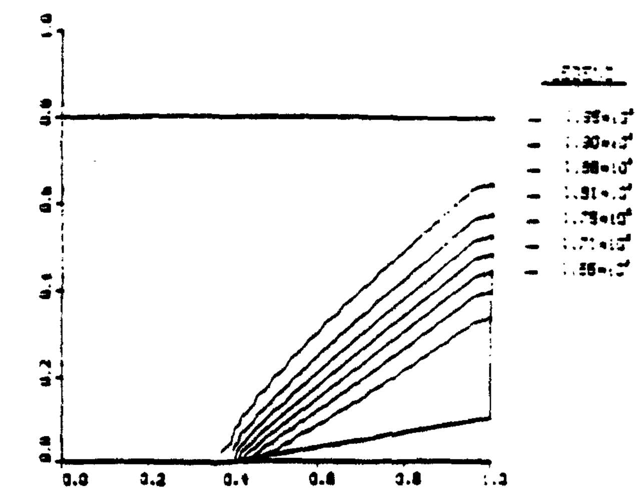


Figure 7a First order Godunov solution for the coarse grid shown in Figure 6.

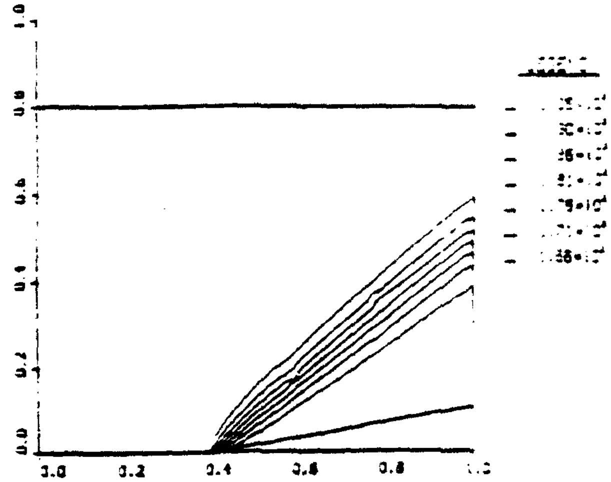


Figure 7b Second order Godunov solution for the grid shown in Figure 6.

To improve the accuracy a higher grid density is required in the region of discontinuity. This is achieved by subdividing the original elements of the grid in regions of large changes in flow parameters.

A variety of criteria can be devised to identify regions which require mesh refinement. An example is given in Ref. 2 where a preset condition is imposed on the resolution from local derivatives of the flow parameters. The implementation of this criteria in FUGGS would have led to a significant loss of computational efficiency because the stencil for the Laplacian is nonlocal and would require more than one level of indirectness. Instead we used a simple parameter variation criteria based on the local variation in pressure or density to select the grid regions needing refinement. Figure 8 shows an enhanced grid derived from the mesh shown in Figure 6 by two levels of subdivision. The number of triangles in this case increase from 800 to 1200. Figure 9 shows isomach lines of the solution using the second order method for the same shock on wedge problem as in Figures 7a and 7b. The improvement in shock resolution is dramatically noticeable. This problem also illustrates the ability of unstructured grid methods to provide local resolution for important flow features, without requiring excessive overhead for other regions of the computational domain.

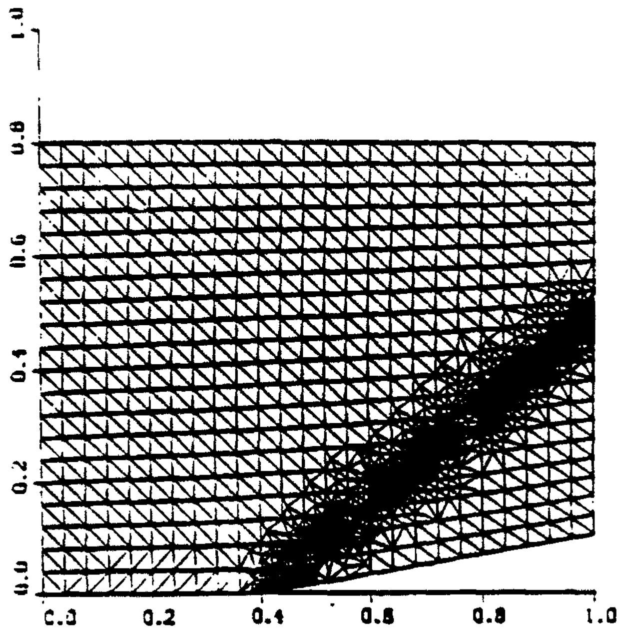


Figure 8 Improved grid for the shock on wedge problem with two levels of refinement based on 5% variation in local value.

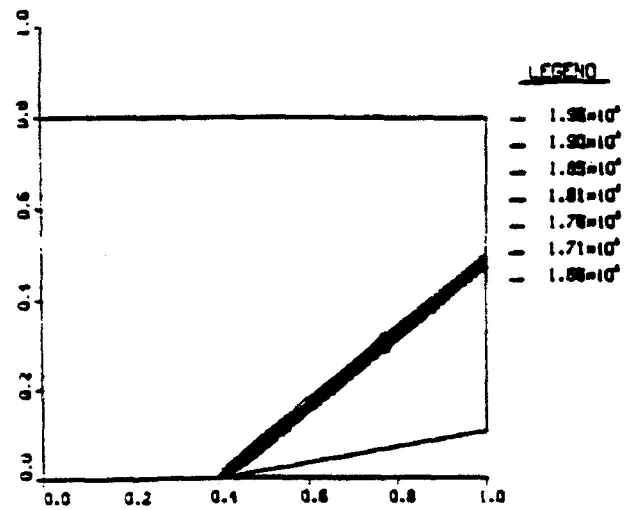


Figure 9 Second order Godunov solution for the shock on wedge problem using a grid with two levels of refinement.

c. Subsonic Flow

A challenging test problem to assess the performance of Euler codes for subsonic flow has been suggested by Pulliam [19]. He has computationally simulated a steady subsonic flow over an ellipse with major to minor axis ratio of 6:1. The numerical solution of Euler equations reported for this case at $M_\infty = 0.2$ with angle of attack $\alpha = 5°$ produced a lift coefficient of $C_L = 1.545$. As is well known from D'Alembert's Paradox, inviscid flow at low Mach numbers should yield $C_L = 0$ and have zero drag for a profile of an arbitrary shape. For this reason the problem posed by Pulliam is a good indicator of the accuracy and amount of artificial dissipation introduced by a numerical algorithm. Moreover, while a Euler solver is not meant to treat potential flow, a general purpose solver should be capable of simulating such flow conditions if they occur in a portion of a given problem without resorting to a different algorithm. In making a transition to full Navier-Stokes treatment, the use of a Euler solver is an essential step; it is important to have confidence that the artificial viscosity introduced does not dominate the solution.

For the case under consideration, it is very important to understand in detail the potential flow solution over an ellipse. Fortunately, the analytical solution is available and is relatively simple. The complex potential for the flow over a cylindrical ellipse is given by the following [20]:

$$F(z) = -\frac{1}{2} M_\infty (a+b) e^{-i\alpha} \left[\frac{z + \sqrt{z^2 - (a^2 - b^2)}}{a+b} + \frac{z - \sqrt{z^2 - (a^2 - b^2)}}{a-b} \right]$$

where $Z = x+iy$ and M_∞ is the Mach number. By taking the gradient of the potential we can find the velocity flow field explicitly:

$$\frac{U}{U_\infty} = \frac{(1+\lambda)\sin(\theta+\alpha)\sin\theta}{\lambda^2\cos^2\theta + \sin^2\theta}, \qquad \frac{V}{U_\infty} = \frac{(1+\lambda)\lambda\sin(\theta+\alpha)\cos\theta}{\lambda^2\cos^2\theta + \sin^2\theta}, \tag{3}$$

where $\lambda = b/a$ is the ratio of minor to major axis, θ is the angle in polar coordinates from the center of the ellipse, and α is the angle of attack.

In examining this equation, we find that the maximum value of velocity is a strong function of λ. For an ellipse with $\lambda = 1/6$, the maximum value V/U_∞ occurs at $\theta = 0$ or π and where $V_{MAX}/U_\infty = 7\sin\alpha$. For a flat plate where $\lambda \to 0$ the maximum velocity is infinite. The angle α defines the distance between the stagnation point where the velocity is zero (at $\theta = -\alpha$ and $\pi-\alpha$) and the point where the velocity is maximum (at $\theta = 0$ and π). For the case selected by Pulliam the distance between the point with minimum and maximum velocity is 0.19% of the length of the major axis.

This means that the gradient of velocity along the major axis of the ellipse in the vicinity of stagnation points is extremely high. With ~ 1000 points uniformly distributed on the surface of the ellipse, only one grid spacing is available to resolve both the stagnation point and the point at which maximum velocity occurs. Even though one would normally construct a nonuniform grid in the vicinity of the stagnation point, we estimate that enormous computational resources would be required to resolve the characteristic scale length for this problem. Traditional methods encounter difficulties in this situation because spatial splitting leads to a poor estimate of the gradient and

the low connectivity of the mesh introduces spurious vorticity.

We performed two simulations for the conditions described by Pulliam. The number of nodes used on the surface of the ellipse is the same as in Ref. [19]. The grid is shown in Figure 10 for these simulations in the region immediately proximate to the ellipse. This grid is of poor quality and highly distorted; contains ~ 6000 vertices and ~ 130 points on the surface of the ellipse. The results are shown in the form of pressure contours in Figures 11 and 12 for the first order and second order solvers respectively. In the case of the first order algorithm, we obtained a lift coefficient of $C_L = 0.29$. The pressure contours for this simulation are not smooth, attributable to the low level of accuracy of the solver. The same situation resulted in $C_L = 0.252$ when computed with the second order solver, and as can be seen in Figure 12 the pressure contours are considerably smoother. The result presented by Pulliam was $C_L \equiv 1.55$, almost an order of magnitude higher than achieved with FUGGS. This highlights an important quality of our approach: the low generation of artificial viscosity. In comparison the lift obtained by Pulliam is as high as one would expect from thin profile theory and hence would mask real viscosity effects if they were added to the algorithm.

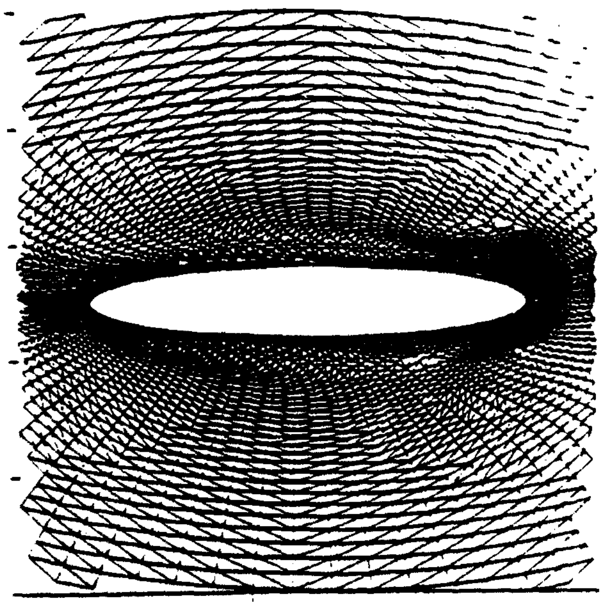


Figure 10 Section of the grid used in simulation of subsonic flow over an ellipse for conditions suggested by Pulliam [19].

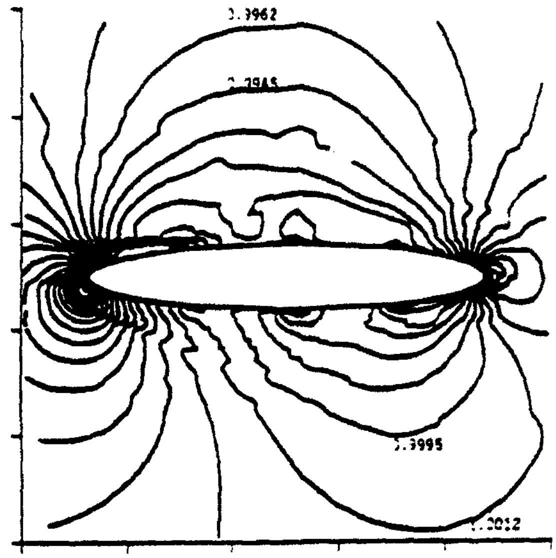


Figure 11 First Order Euler Solution for 6:1 Ellipse. Pressure contours. $\alpha = 5°$; Mach = 0.2; 6065 vertices; $C_L = 0.381$; $C_D = 0.101$.

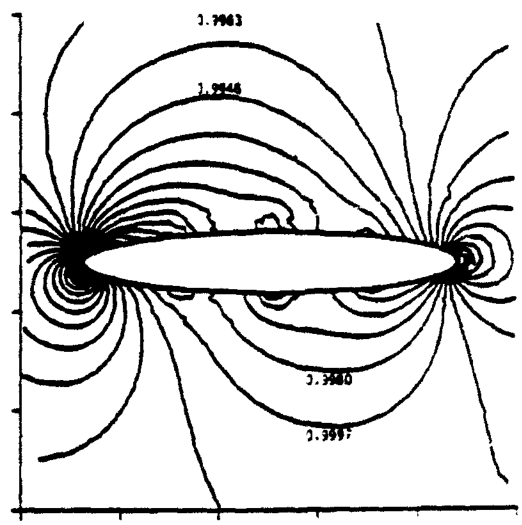


Figure 12 Second Order Euler Solution for 6:1 Ellipse. Pressure Contours. $\alpha = 5°$; Mach = 0.2; 6065 vertices; $C_L = 0.252$; $CD = 0.004$.

We also simulated flow over a cylinder at M = 0.2. The grid for this case is shown in Figure 13. We examined the numerically produced lift with inflow conditions at various angles with respect to the x - axis (0°, 5°, 20°, 45°). The lift coefficient was angle independent and had a value $C_L = 0.76$, almost 20 times smaller than reported by Pulliam, whose results are angle sensitive. With the first order scheme we achieved a lift coefficient of $C_L = 0.47$ with the drag coefficient $C_D \equiv 1.49$. For the second order scheme, shown in Figure 14, the drag coefficient was reduced to $C_D = 0.19$ but the lift coefficient increased somewhat to the

value cited above. We also investigated the effects of grid refinement and found that a simple one level of refinement (adding ~ 400 vertices) led to a modest reduction in lift coefficient of about 20%. To reinforce a point made earlier, all of the results were achieved with no "free" parameters to adjust. These parameters are present in many CFD codes in the form of coefficient for artificial viscosity terms present the practitioner with a practical problem of how they should be selected for different flow conditions.

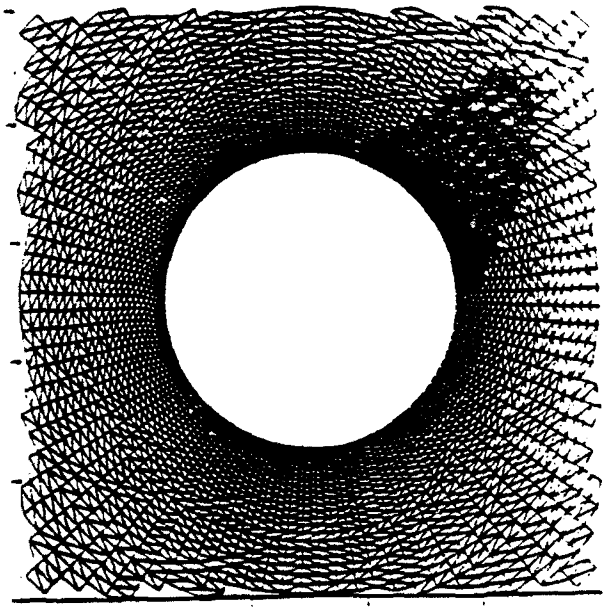


Figure 13 Grid for simulation of flow over a cylinder at varying inflow angles with respect to the mesh.

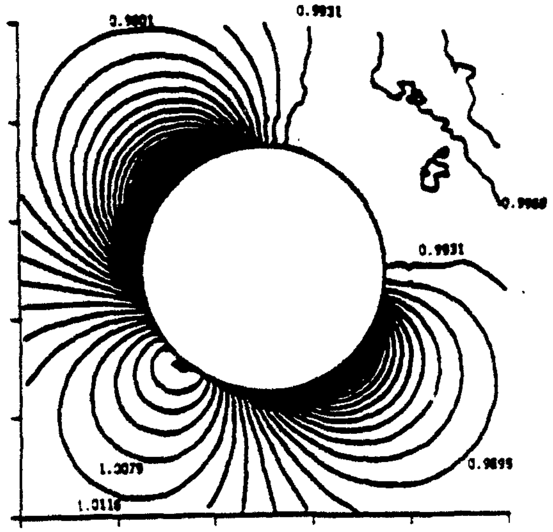


Figure 14 Second Order Euler Solution for a circular cylinder: $\alpha = 45°$; Mach = 0.2; 6311 vertices; $C_L = 0.761$; $C_D = 0.196$.

d. Hypersonic Flow

To demonstrate the versatility of the method for the entire range of flow regimes we have simulated a hypersonic flow test problem. One of the advantages of the Godunov type methods is that for the whole range of calculations performed (from low subsonic flow, supersonic flow, unsteady flow with strong shock, or hypersonic flow at Mach number M=32) it is unnecessary to change or adjust the numerical algorithm. In Ref. 21 performance of first and second order Godunov methods has been analyzed for hypersonic flow regimes. There, as a test problem, an analytical solution was used for a hypersonic flow around a flat plate of finite thickness. This solution was obtained based on the analogy between hypersonic flow over a flat plate of finite thickness and a strong planar explosion. Here we will use one expression from Ref. 21 which defines the shape of the shock wave as a function of plate thickness d, γ the adiabatic coefficient, and α a nondimensional scale factor related to the energy released at the stagnation point.

$$Y_{SHOCK} = \left(\frac{1}{2} D_f \frac{dx^2}{2}\right)^{1/3}$$

where D_f is a coefficient of order unity,

$$a = k_1 (\gamma-1)^{k_2 + k_3 \ln(\gamma-1)}$$

while $k_1 = 0.36011$, $k_2 = -1.2537$, and $k_3 = -0.1847$.

As a direct comparison we solved the hypersonic flow problem for the same set of conditions as in Ref. 21:

$$U_\infty = 10011 \text{ meters/sec}, \ p = 98.72 \text{ Pa},$$
$$\rho = 1.24\times10^{-3} \text{kg/m}^3 \text{ and } \gamma = 1.2.$$

The grid used for this simulation is shown in Figure 15. This grid has ~ 5500 vertices and it's spatial resolution at the leading edge of the plate is of the same order as that of a 300 x 60 rectangular grid used in Ref. 12.

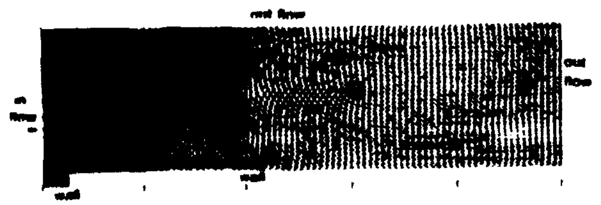


Figure 15 Grid for simulation of hypersonic flow over a flate plate.

In Figure 16 results for this simulation are shown in the form of pressure contours. Figure 16 also shows the location of the analytically calculated shock front by a discrete line (squares). The shock resolution and accuracy of its location are comparable to that obtained in Ref. 21, even

though our triangular grid has less than 1/3 the nodes than the rectangular grid used in Ref. 21. This is due to the fact that in constructing the triangular grid we had the flexibility to place the highest concentration of nodes in the area of the leading edge where the main properties of the flow are established.

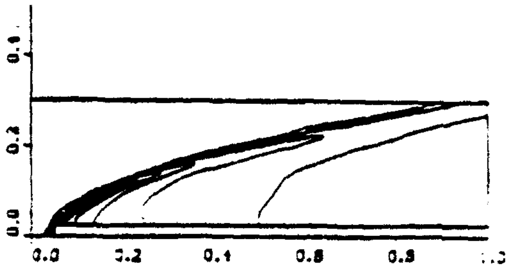


Figure 16 Second order solution for a flat plate. Pressure Contours. Mach = 32; 5509 grid vertices; $P_{max} = 5.0x10^4$; $P_{min} = 98.7\ P_a$.

CODES COMPUTATIONAL EFFICIENCY

During the code development effort, great attention was paid to the code data structure, its efficiency and extendability to three dimensional calculations. In fact, the two dimensional version of the code has all the data structures required for the three dimensional simulations. That fact should be factored in comparing our storage overhead figures to those in other codes. Also while developing FUGGS we made a decision not to rely on machine-dependent functions, in order to assure portability. This feature is very important in the current supercomputing environment where a host of powerful parallel supercomputers and super workstations with diverse architecture are available and useful for different aspects of design.

The following performance characteristics have been achieved for the latest version of the FUGGS code:

1. First Order Godunov version:

Memory Requirement	36 places per triangle includes 5 physical quantities integer indexing arrays all geometric parameters
CPU Performance CRAY XMP-24 STELLAR 1000	 15 µsec/vertex/timestep 79 µsec/vertex/timestep

2. Second Order Godunov version:

Memory Requirement	39 places per triangle
CPU Performance CRAY XMP-24	45 µsec/vertex/timestep Monotonicity step 50% Characteristic limiter 15% Riemann Solver 30% Integration 5%
STELLAR 1000	214 µsec/vertex/timestep

These numbers are provisional since the code is still under development. We feel that further improvements in code performance will be achieved with respect to both timing and storage requirements.

CONCLUSION AND DISCUSSION

We have presented a method for the numerical solution of Euler equations on an unstructured triangular grid. The method was tested for a wide range of flow conditions from low subsonic flow and unsteady flow with strong shock waves to hypersonic flow with Mach 32. For all these regimes, the method performed extremely well both in terms of solution accuracy and computational efficiency. The method is very robust and does not resort to adjustable computational parameters for the tested range of flow conditions. This is due to the fact that the numerical algorithm in FUGGS is based on Second Order Godunov schemes adapted to triangular grids. The method appears natural for unstructured triangular grids because the greater connectivity intuitively should lead to greater accuracy in eliminating errors introduced by splitting. In a typical hexagonal (or greater) control volume the contribution of fluxes is available from all six adjacent directions as opposed to just two in the case of a rectangular grid. Since the FUGGS method has been implemented on unstructured grids, it is possible to simulate flows over bodies of arbitrary geometry where the grid density can be concentrated in a region of flow discontinuity.

Especially interesting is the code's superior performance for the simulations of subsonic flow. For the test cases calculated here, our method appears to perform better than the leading industry codes like ARC2D and SYMTVD. We think that the two main reasons for the better performance are multidirectional splitting (to distinguish from two directional splitting typical for logically structured quadrilateral grids) and finite volume integration, more should be done to investigate this important aspect of the code's performance.

Historically, Euler solvers were developed to simulate nonisoentropic flows for which potential flow assumptions are incorrect. From the original development of numerical methods for the solution of Euler's equations, great effort has been devoted to resolving shocks and contact discontinuities, producing in dramatically improved results for shock wave hydrodynamics. At the same time, attention to the accurate solution of the velocity gradients has been neglected. While these gradients are more difficult to discern than shock waves, they are more prevalent in practical flow

problems and could lead to very significant errors in such important parameters as lift and drag coefficients. In addition, all vorticity and viscosity dominated phenomena depend on accurate solution of the velocity gradients. In view of the performed numerical simulations for subsonic flow over the ellipse and cylinder it is clear that unless these features are resolved, the numerical solution of Euler equations can introduce spurious vorticity, making the results from a full Navier-Stokes implementation impossible.

Acknowledgment

This work was partially supported by Wright-Patterson Research and Development Center under contract No. F33615-88-C-3002 and by DARPA through AFOSR contract No. FY9620-89-C-0087 We would also like to thank Dale Nielsen, Jr. of Lawrence Livermore National Laboratory for the initial gridding software and STELLAR Computer for providing assistance and facilities.

REFERENCES

1. F. Angbrand, V. Boulard, A. Dervieux, J. Periaux, and G. Vuayasundaram, in *Computing Methods in Applied Sciences and Engineering*, edited by R. Glowinski et al. (North-Holland, Amsterdam, 1984).
2. F. Angbrand, V. Boulard, A. Dervieux, J.A. Desideri, J. Periaux, and B. Stouggiet, in *Proceedings Ninth International Conference on Numerical Methods in Fluid Dynamics, Saclay, France*, 1984, edited by Soubbaramayer and Boujot, Lecture Notes in Physics, Vol. 218 (Springer-Verlag, 1985).
3. L. Fezoui and B. Stouggiet, "A Class of Implicit Upwind Schemes for Euler Simulations with Unstructured Meshes," J. of Comp. Phys. 84, 174-206 (1989).
4. R.A. Shapiro and E.M. Murman, "Adaptive Finite Element Methods for the Euler Equations," AIAA-88-0034, 26th Aerospace Sciences Meeting, Reno, Nevada, 1988.
5. R. Lohner, K. Morgan, and D.C. Zienkiewicz, "Finite Element Methods for High Speed Flows," AIAA *7th Computational Fluid Dynamics Conference*, Cincinnati, Ohio, AIAA Paper 85-1531 (1985).
6. R. Lohner and K. Morgan, "Improved Adaptive Refinement Strategies for Finite Element Aerodynamic Computations," AIAA *29th Aerospace Sciences Meeting*, Reno, Nevada, AIAA Paper 86-0499 (1986).
7. D. Mavriplis and A. Jameson, "Multigrid Solution of the Two-Dimensional Euler Equations on Unstructured Triangular Meshes, AIAA-87-0353, 1987.
8. T.K. Dukowicz, M.C. Cline, and F.L. Addessio, "A General Topology Godnuov Method," J. of Comp. Phys. 82, 29-63 (1989).
9. T.J. Baker and A. Jameson, "A Novel Finite Element Method for the Calculation of Inviscid Flow Over a Complete Aircraft," *Sixth International Symposium on Finite Element Methods in Flow Problemns*, Antibes, France (1986).
10. A. Jameson and T.J. Baker, "Improvements to the Aircraft Euler Method," AIAA *25th Aerospace Sciences Meeting*, Reno. Nevada, AIAA Paper 87-0452 (1987).
11. T.J. Baker, "Developments and Trends in Three-Dimensional Mesh Generations," *Transonic Symposium*, NASA Langley Research Center, Virginia (1988).
12. A. Jameson, T.J. Baker, and N.P. Weatherill, "Calculation of Inviscid Transonic Flow Over a Complete Aircraft," AIAA *24th Aerospace Sciences Meeting*, Reno, Nevada, AIAA Paper 86-0103 (1986).
13. L. Greengard and V. Rokhlin, "A Fast Algorithm for Particle Simulations," *J. Comp. Phys.* 73, 325-348 (1987).
14. S. Eidelman, P. Collela, and R.P. Shreeve, "Application of the Godunov Method and It's Second Order Extension to Cascade Flow Modeling," *AIAA Journal*, v. 22, 10 (1984).
15. B. van Leer, "Towards the Ultimate Conservative Difference Scheme, V.A. Second Order Sequel to Godunov's Method," *J. Comp. Phys.* v. 32, 101-136 (1979).
16. P. Collela and P. Woodward, "The Piecewise Parabolic Method (PPM) for Gasdybanucak Simulations," *J. Comp. Phys.* v. 54, 174-201 (1984).
17. T.J. Barth and D.C. Jespersen, "The Design and Application of Upwind Schemes on Unstructured Meshes," *27th Aerospace Sciences Meeting*, AIAA-89-0366, Reno, Nevada (1989)
18. H.M. Glaz, P. Collela, I.I. Glass, and R.L. Deschambault, "A Detailed Numerical, Graphical, and Experimental Study of Oblique Shock Wave Reflections," DNA-TR-86-365, Technical Report, 1986.
19. T.K. Pulliam, "Computational Challenge: Euler Solution for Ellipses," AIAA-89-0469, Reno, Nevada, 1989.
20. Schaum's Outline Series, Fluid Dynamics, by W.F. Hughes and J.A. Brighton, McCraw-Hill Book Co., New York, 1967.
21. S. Eidelman, "Application of the Hypersonic Analogy for Validation of Numerical Simulations," AIAA Journal 27, 11, 1566-1571 (1989).

Reflection of the Triple Point of the Mach Reflection in a Planar and Axisymmetric Converging Channels

Shmuel Eidelman and Itzhak Lottati

Science Applications International Corporation
McLean, Virginia, USA

9th Mach Reflection Symposium
Freiburg
June 1990

Reflection of the Triple Point of the Mach Reflection in Planar and Axisymmetric Converging Channels

Shmuel Eidelman and Itzhak Lottati

Science Applications International Corporation,
McLean, VA. USA

Introduction

Depending on their parameters, the encounter between a planar shock wave and a wedge can produce a classic case of the Mach Reflection. The Mach Reflection has a characteristic triple point, where three shocks and the contact discontinuity coalesce. In a shock tube or in a channel, a developed Mach Reflection can reflect further from the walls opposite the wedge. In this case, the Mach shock of the Mach Reflection will start reflecting when its triple point reaches the wall opposite the wedge. Upon reflection of this shock wave, a secondary Mach Reflection can form. Although the primary Mach Reflection has received considerable attention in scientific literature (Refs. 1,2,3), the phenomenology of the subsequent reflections was gone virtually unnoticed. In our literature review, we found only a short qualitative description of the phenomena by Bazhenova and Gvozdeva (Ref. 4). This omission is unfortunate, since it is a very practical case for propagation of the shock waves in channels of variable cross sections.

The direct simulation of the various cases of Mach Reflection has only become possible in the last decade. This problem is a challenging test for the numerical methods used in Computational Fluid Dynamics (CFD). An impressive demonstration of the capabilities of the direct numerical simulations of Mach Reflection phenomena is given by Glaz et al. (Ref. 5). They demonstrate that all the important phenomenology of the Mach Reflection, including slip line vortex and Mach shock wave bulging, can be simulated directly. This was achieved by using the Second Order Godunov method, numerical technique, developed in 80th, which is extremely robust and allows very accurate simulation of flow discontinuities.

The Second Order Godunov method was implemented on rectangular grids (Ref. 6) and in a few cases on general quadrilateral grids (Ref. 7). This approach has limited application, since the structured quadrilateral grids have great difficulty describing a complicated computational domain with multiple bodies of different geometries and scales. Recently, we have implemented the Second Order Godunov for unstructured triangular grids (Ref. 8). This enables us to combine the robust and accurate numerical algorithm with a griding technique, allowing us to describe very complex domains with ease and efficiency. In addition, we have developed a novel Dynamic Grid Adaptation methodology which allocates a dense computational grid only to regions where

enhanced resolution is needed to resolve strong gradients in flow parameters. As demonstrated in our paper, this enables an extremely economical allocation of computational resources and accurate simulation of a complicated phenomena like Mach Reflection.

In our study, we numerically simulate the formation of a Double Mach Reflection on a sloped wall of a converging channel, with subsequent reflection of the reflected wave at the straight wall of the channel. Presented here numerical results were obtained with the new numerical technique and we will describe in detail all the important new elements which we have introduced.

The Problem

Figure 1 shows a converging channel with a sloped wall at 27°. The figure illustrates our assumption that a Mach 8.7 shock wave travelling normally to the parallel walls enters the channel at the left hand side. According to analysis presented in Reference 9, this shock will have a Complex Mach Reflection when it encounters the converging wall of the channel. At some stage of the reflection process, the triple point will reach the opposite wall of the channel. Here the Mach stem shock wave will become incident, moving at an angle to the channel wall, as illustrated in Figure 2. The shock and wedge parameters chosen in our problem will cause formation of a secondary Mach reflection. The question is: What form will this secondary reflection take? Bazhenova and Gvozdeva offer a very general description of the anticipated effect, illustrated in Figure 3 (Ref. 4). In this reference, a system of secondary reflections shows the incident and Mach shocks are interchanging their positions with every new reflection, and the strength of the shock waves is increasing. It is not clear from Reference 4 what type of Mach Reflection will form, or how the secondary reflected wave, which expands in already perturbed gas, will be affected by the interactions with the strong slip surfaces located behind the original Mach shock.

We will directly simulate formation of the Mach Reflection at the channel oblique wall, as well as all secondary reflections which will occur according to the conditions outlined above for the channel geometry shown in Figure 1. In addition we will consider cases in which the channel shown in Figure 1 is axisymmetric and will study the same problem for this case. The motivation is further study of the phenomenology of shock wave focusing when a three-dimensional contraction occurs.

In our study we will consider an ideal, invisid gas which can lead to some distortion of our results compared with experimental data. However, we believe that this simplification will still capture the main phenomenology of wave formation and reflection, and will be of general value to the Mach Reflection Theory.

Numerical Method

In Reference 8 we introduced a new numerical algorithm: FUGGS (Fast Unstructured Second Order Godunov Solver), for solving Euler's equations of gasdynamics on unstructured triangular grids. The algorithm formulated and tested in Reference 8 is vertex-based. Here we will describe a new volume based version of the FUGGS method. The new version of the algorithm as illustrated in our paper, produces considerably more accurate solutions and it is more efficient. This contradicts published results (Ref. 10,11) on implementation of the triangle-based TVD schemes for unstructured triangular grids. The new algorithm has been validated for the range of subsonic, supersonic and hypersonic steady state and transient problems. Here we show only results for Mach Reflection in planar and axisymmetric channels..

The new triangle-based version of the FUGGS algorithm was extended to allow dynamic adaptive grid refinement for transient problems. We will give a description of the dynamic grid adaptation methodology used in FUGGS code.

A three dimensional version of the FUGGS algorithm was developed in an extremely short period of time. This was made possible by the simple structure of the basic algorithm. We will not present simulation results for the three dimensional FUGGS, however, the main elements of the FUGGS algorithm implementation in the three dimensions will be illustrated.

Vertex-Based and Triangle-Based Integration Algorithms

We consider a system of Euler equations written in conservation law form in three dimensions as:

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{f}}{\partial x} + \frac{\partial \mathbf{g}}{\partial y} + \frac{\partial \mathbf{h}}{\partial z} = 0 \tag{1}$$

where

$$\mathbf{U} = \begin{vmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{vmatrix}, \ \mathbf{f} = \begin{vmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ \rho uH \end{vmatrix}, \ \mathbf{g} = \begin{vmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vw \\ \rho vH \end{vmatrix}, \ \mathbf{h} = \begin{vmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + p \\ \rho wH \end{vmatrix}$$

Here u, v, and w are the x, y, and z velocity vector components, p is the pressure, ρ is the density and H is the total enthalpy and E is total energy of the fluid. It is assumed that a mixed (initial conditions, boundary conditions) problem is properly posed for the set of equations (1), that an initial distribution of the fluid parameters is given at t=0, and the boundary conditions defining a unique solution are specified for the computational domain.

We seek a solution of the system of equations (1) on the computational domain which is decomposed into tetrahedrons (triangles in two dimensions) with arbitrary connectivity. An overwhelming advantage of this method of domain decomposition is the ability to resolve extremely complicated geometries and flow regimes accurately and efficiently. This has been demonstrated in numerous publications on this topic (Ref. 12, 13, 14).

There are several options possible for storing natural physical parameters of the problem on an unstructured tetrahedral or triangular grid. In particular, we have examined: i) vertex centered; and ii) tetrahedron (or triangle) centered. These two approaches, while equivalent from the point of view of the formal numerical representation of the governing equations, lead to different algorithms. As shown below, this will have important consequences not only on data structure and algorithm efficiency, but moreover the different connectivity will affect the overall accuracy of the numerical solution.

In Figure 4, a fragment of the two-dimensional computational domain is shown. Here, together with the original triangular grid (solid lines), the secondary grid (broken line) is shown. This secondary grid is formed by connecting the barycenters of the primary grid. If a vertex based grid is used, the physical parameters of the problem are stored at vertices A, B, C..., and the integration is done for the volumes delineated by the polygons of the secondary grid. For instance, integration volume associated with vertex A is defined by the edges ab, bc, cd, de, ef, fa. For a triangle-based grid the physical parameters will be stored at the nodes of the secondary grid, and integration volume will be the triangle itself. We have shown (Ref. 15) that these two approaches lead to numerical algorithms with different connectivity, accuracy and efficiency. The fundamental algorithm of the second order Godunov method implemented in FUGGS can be illustrated in two dimensions for an edge of the grids control volume shown in Figure 5. The algorithmic steps of the second order Godunov method can be defined as follows:

1. Find the value of the gradient at the vertex point (or at the baricenter of the triangle for the triangle-based version) for the gasdynamic Parameter U;

2. Using the gradient values, find the interpolated values of U at the edges defining the control volume (sides of the triangle for the triangle-based scheme)

3. Limit these interpolated values based on a monotonicity condition (Ref. 16)

4. Subject the resulting values to the characteristic's constraints (Ref. 6)

5. Solve the Riemann problem for the corrected values.

This last step completes the definition of the fluxes at the edges of the control

volume. The flux values can be stored at the edges and the flux calculation loop will be arranged for the list of edges, which is the largest vector in the system. If the algorithm is vertex-based to calculate U^{n+1} values, we will integrate the fluxes at the edges of the secondary grid which define the control volume for the vertex. For the triangle-based algorithm U^{n+1}, value is obtained by integrating the fluxes at the sides of the triangles.

Implementation of the algorithm in three dimensions will have the same basic steps in flux calculation 1-5. To illustrate that point, Figure 6 shows a tetrahedral element of the grid. Here the fluxes are defined on the faces of the tetrahedral at the edge points. At step 1 the gradient is caluclated at the barycenter cell point for the tetrahedral. All the rest of the steps are identical to those described above. To find the value of U^{n+1} in the three dimensional case, we will add fluxes defined at the faces of the tethraletral. Most elements developed for the two dimensional code are applicable to this implementation of the three dimensional algorithm.

Direct Dynamic Refinement Method (DDRM)

Practical numerical simulations of the fluid dynamic problems call for modeling flows over complicated shapes. In addition, important flow features such as shed vortices, shock waves, slip lines and boundary layers usually have widely varied lengths and time scales and need to be resolved. Accurate solution of these problems require computational grids dynamically adapted to the evolving flow feature, and with full control over solution accuracy in the key regions of the computational domain. It is commonly accepted that only unstructured grids can provide full flexibility in obtaining the local grid resolution sufficient to accurately resolve subscale flow features. The five years since the introduction of these grids and methods in CFD research have produced landmark simulations clearly demonstrating their advantages (Ref. 12, 14, 17).

Although a number of research groups have demonstrated application of unstructured grids to simulations of steady state problems (Ref. 14, 17, 18), simulations of time-dependent problems were accomplished by a significantly smaller group (Ref. 19, 20). An adaptive refinement method developed by Lohner (Ref. 20) is based on a hierarchical system of grid refinement/coarsening in which each level of refinement has six possible cases and coarsening three cases of triangular cells formation. Every layer of refinement has a father/son relation with the previous layer, and all these layers of refined mesh move on the basic predefined grid. This technique has the demonstrated capability of carrying out simulations of extremely complex flow regimes. However, its rigid hierarchic approach to generating grid results in some implicit limitations. For example, a dynamically evolving grid will not have an element larger than the cell of the initial grid, or it will be impossible to reduce the cell volume abruptly in some areas without passing through all the necessary level of refinement.

In our paper we will report a new method of dynamic grid adaptation. This method is based on direct refinement and reconnection in the areas of monotonic flow preceding the regions with strong flow gradients. In Figure 7 we have illustrated the basic process of refinement accomplished in the DDRM method. The original grid is shown in Figure 7a. Figure 7b illustrates a one step grid refinement in which a new vertex is introduced into a triangular cell forming three new cells. This is followed by reconnection which modifies the grid in a manner demonstrated in Figure 7c. The process of refinement and reconnection can be continued until the necessaary grid resolution is achieved, as illustrated in Figures 7d and 7e. This direct approach to the grid refinement grants extreme flexibility in resolving local flow features. A similar simple method is applied to grid coarsening. In the first step of coarsening the marked vertices, all associated elements of the grid are simply removed, as shown in Figure 8a. During the second step, this void in the grid is filled with new larger triangles (Figure 8b), and then reconnected as shown in Figure 8c.

The Direct Dynamic Refinement Method (DDRM) was implemented for the second order-Godunov method (FUGGS algorithm Ref. 7, 15). Here we demonstrate its performance for a classical Mach Reflection problem.

Results

In Figures 8a, 8b, and 8c, simulation results are shown in the form of density contours for different stages of Mach Reflection in a planar channel. To illustrate the dynamics adaptation of the computational grid to the solution in the same figures, we show the grid as it evolves in time. The numerical solution develops as a classical case of Mach Reflection. Because we have assumed that the gas is ideal with $\gamma = 1.4$, according to Ben-Dor and Glass (Ref. 9) for the shock wave and wedge angle conditions chosen we should have a case of Complex Mach Reflection (CMR). We can observe in Figures 8a, 8b and 8c that the density contours definitely display the pattern of discontinuities attributed to CMR. In these figures we observe a well defined slip line vortex, slip line, triple point and the kink. For real gas, in this case, the Double Mach Reflection should occur. The slip line and slip line vortex will be located close to the Mach shock and will cause the Mach shock bulging (Ref. 5). However the perfect gas assumption will lead to CMR and extensive bulging will not arise, as is accurately predicted in our simulations. It is striking to observe in Figures 8a, 8b, and 8c that the numerical grid closely follows the evolving system of waves, and the high density grid is only observed in the areas of shock waves, slip lines and other flow discontinuities. The result is tremendous savings in both CPU and storage. For example, the grid shown in Figure 8c has only 6000 points (an equivalent of a grid 60x100 in the case of a structured rectangular grid).

Reflection of Mach shock from the wall opposite the wedge will start immediately after the stage shown in Figure 8c. This reflection results in formation of the secondary Mach Reflection which expands towards the channel's oblique wall. In Figure 9a, this secondary Mach Reflection can be clearly identified. In Figure 9b, the blow up of the region of the secondary Mach Reflection is shown. All the distinguishing characteristics of the Mach Reflection can be identified in this figure, including triple point, Mach shock, reflected shock and slip line. In addition to all these features, the secondary Mach Reflection has an additional kink, resulting from interaction of the reflected shock with the slip surface. It is clear that this interaction will affect significantly the dynamics of the secondary reflection.

In Figures 10a, 10b, and 10c, simulation results are shown for the Mach Reflection in an axisymmetric channel which has the same cross section as the planar channel. For direct comparison here the simulation results are presented in the same format as in Figures 8a, 8b, and 8c for the case of a Mach Reflection in a planar channel. The Mach Reflection in Figure 10a is analogous to its planar counterpart in Figures 8a and 8b. In Figure 10b it can be observed that the area of the shock between the triple point and the kink in the reflected shock tilts towards the axis of symmetry of the channel. This is even more pronounced in Figure 10c where the density contours are shown before the secondary reflection starts. It is apparent that the secondary reflection of the Mach shock will occur earlier in the axisymmetric channel than in its planar counterpart. Contraction in the radial direction results in a significant jump in density upon reflection. In Figure 11a we see that at the initial stages of the axisymmetric reflection, maximum density increased three-fold compared with the values observed in the initial reflection. This increase in density affects the increment between the contour levels displayed in Figure 11a and causes the slip line not to show. In Figure 11c a more advanced stage of the secondary reflection is shown. To examine in more detail the features of the secondary reflection in Figures 11b and 11d, we show an enlarged view of the secondary reflection region corresponding to Figures 11a and 11c. In these figures, we can observe the formation of a distinct reflected wave pattern with a characteristic double kink of the reflected wave similar to that seen in the secondary reflection in a planar channel. In the axisymmetric case, the secondary reflection is significantly stronger than in the case of a planar channel. Since this reflected wave propagates along the radius of the channel, it will expand rapidly. This can be observed in Figure 11c where the maximum value of density dropped 30% compared with the maximum in Figure 11a. For the same reason the triple point of the secondary Mach Reflection has advanced much farther towards the oblique wall in Figure 11d than in Figure 9b.

Conclusions

A computer code has been developed for Euler's equations of gas dynamics. This code uses unstructured grids for computational domain decomposition and its integration algorithm is based on the Second Order Godunov method. The code uses the Dynamic Grid Adaptation methodology, allowing economical allocation of computer resources to evolving flow features. In turn, it is then possible to carry out accurate simulations of complicated gas dynamic phenomena with affordable computer resources. Here the code has been demonstrated to produce an accurate simulation of Complex Mach Reflection in planar and axisymmetric channels. We also have simulated the initial stages of the secondary Mach Reflection from the channel wall opposite the oblique wall. In this case we observed new wave structures with a characteristic double kink. The formation of the second kink was a result of the interaction between the secondary reflected wave and the original slip line. It was noted that the dynamics of the secondary reflection is different in the planar and axisymmetric cases. In the axisymmetric case reflection in significantly stronger than in the planar case.

References

1. Ben-Dor, G., and I.I. Glass, "Nonstationary Oblique-Shock Wave Reflections: Actual Isopycnics and Numerical Experiments," AIAA J. 16 (1978), pp. 1146-1153.

2. Gvozdeva, L.G., T.V. Bazhenova, O.A. Predvoditeleva, V.P. Fokeev, 1969. Mach Reflection of Shock Waves in Real Gases, Astron. Acta 14:503-8.

3. Hornung, H.G., H. Oertel, R.J. Sandeman, 1979. Transmission to Mach Reflexion of Shock Waves in Steady and Pseudosteady Flow with and without Relaxation. J. Fluid Mech. 90:541-60.

4. Bazlenova, T.V. and L.G. Gvozdeva, "Unsteady Interactions of Shock Waves," Nauka, Moskow, 1977.

5. Glaz, H.M., P. Colella, I.I. Glass, and R.L. Deschambault, A Detailed Numerical, Graphical, and Experimental Study of Oblique Shock Wave Reflections, DNA-TK-86-365, 1986.

6. Woodward, P.R., and P. Colella, "Numerical Simulation of Two-Dimensional Fluid Flow with Strong Shocks," J. Comp. Phys. 54 (1984), pp. 115-173.

7. Eidelman, S., P. Collela, and R.P. Shreeve, "Application of the Godunov Method and Its Second Order Extension to Cascade Flow Modeling," AIAA Journal, v. 22, 10 (1984).

8. Lottati, I., S. Eidelman and A. Drobot, "A Fast Unstructured Grid Second Order Godunov Solver (FUGGS)", 28th Aerospace Sciences Meeting, AIAA-90-0699, Reno, NV, 1990.

9. Ben-Dor, G. and I.I. Glass, 1979 Domains and Boundaries of Nonstationary Oblique Shock-Wave Reflexions: 1. Diatomic gas. J. Fluid Mech. 92:459-96.

10. Barth, T.J. and D.C Jespersen, "The Design and Application of Upwind Schemes on Unstructured Meshes," 27th Aerospace Sciences Meeting, AIAA-89-0366, Reno, NV, (1989).

11. Mavriplis, D. and A. Jameson, "Multigrid Solution of the Two-Dimensional Euler Equations on Unstructured Triangular Meshes," AIAA-87-0353, 1987.

12. Baum, J.D. and R. Lõhner, "Numerical Simulation of Shock- Elevated Box Interaction Using an Adaptive Finite Element Shock Capturing Scheme," AIAA Preprint 89-0653, Presented at the AIAA 27th Aerospace Sciences Meeting, Jan. 8-12, 1989, Reno, NV.

13. Lõhner, R., K. Morgan, J. Peraire and M. Vahdati, "Finite Element Flux-Corrected Transport (FEM-FCT) for the Euler and Navier-Stokes Equations" Chapter 6 in Finite Elements in Fluids Vol. VII (R.H. Gallagher, et al. eds.), J. Wiley and Sons (1988).

14. Mevriplis, D.T., "Accurate Multigrid Solutions of the Euler Equations on Unstructured and Adaptive Meshes," AIAA Journal, 2, V. 28, p. 231, 1990.

15. Eidelman, S. and I. Lottati, "Triangle Based FUGGS and its Validation for Two and Three Dimensional Flow Problems," to be presented at 29th Aerospace Sciences Meeting, Reno, NV, 1991.

16. van Leer, B., "Towards the Ultimate Conservative Difference Scheme, V.A. Second Order Sequel to Godunov's Method," J. Comp. Phys. v. 32, 101-136 (1979).

17. Jameson, A., T.J. Baker and N.P. Weatherill, "Calculations of Inviscid Transonic Flow Over a Complete Aircraft," AIAA 24th Aerospace Sciences Meeting, Reno, NV, AIAA Paper 86-0103, January 1986.

18. Peraire, J., M. Vahdati, K. Morgan and O.C. Zienkiewicz - Adaptive Remeshing for Compressible Flow Computations; J. Comp. Phys. 72, 449-466 (1987).

19. Palmerio, B. and A. Dervieux - Application of a FEM Moving Node Adaptive Method to Accurate Shock Capturing; Proc. First Int. Conf. on Numerical Grid Generation in CFD, Landshut, W. Germany, July 14-17, 1986, Pineridge Press.

20. Lõhner, R. - Adaptive Remeshing for Transient Problems; Comp. Meth. Appl. Mech. Eng. 75, 195-214 (1989).

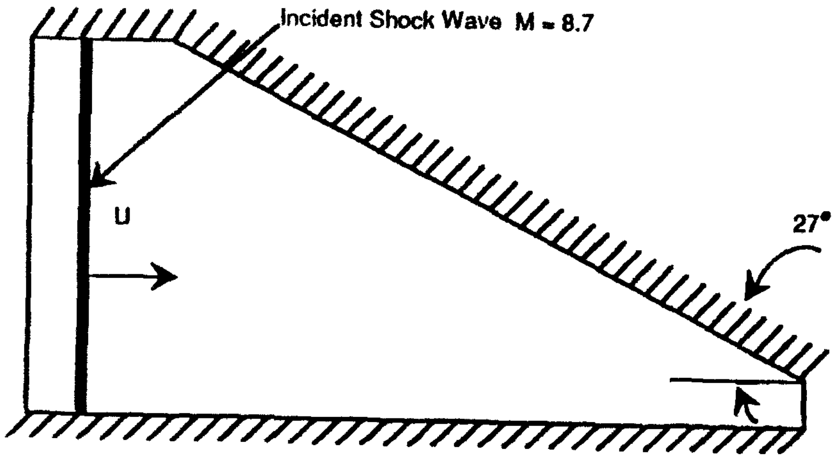


Figure 1. Channel Geometry and Initial Shock Location.

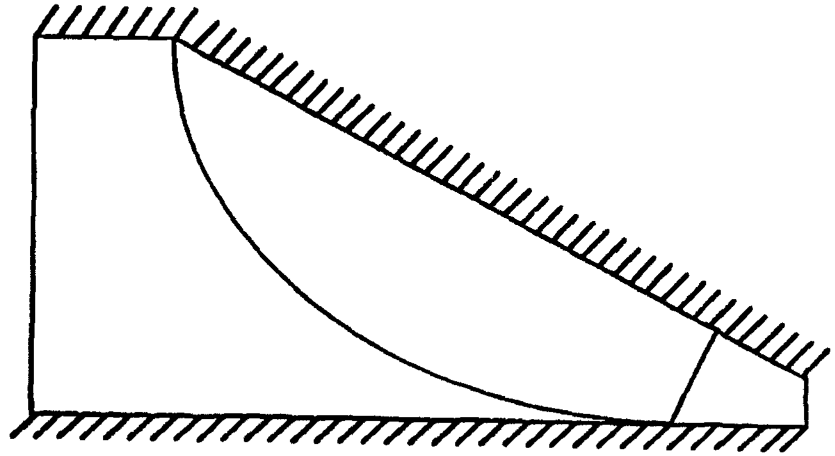


Figure 2. Reflected and Mach Stem Shock Waves at the Start of the Secondary Reflection.

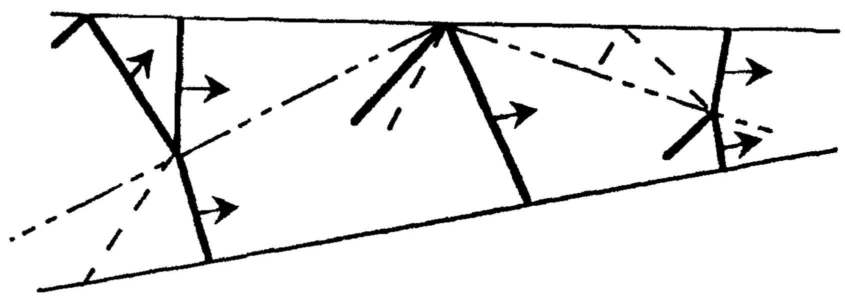


Figure 3. Schematics of Wave Propagation in a converging channel accoding to Bazhenova and Gvozdeva.

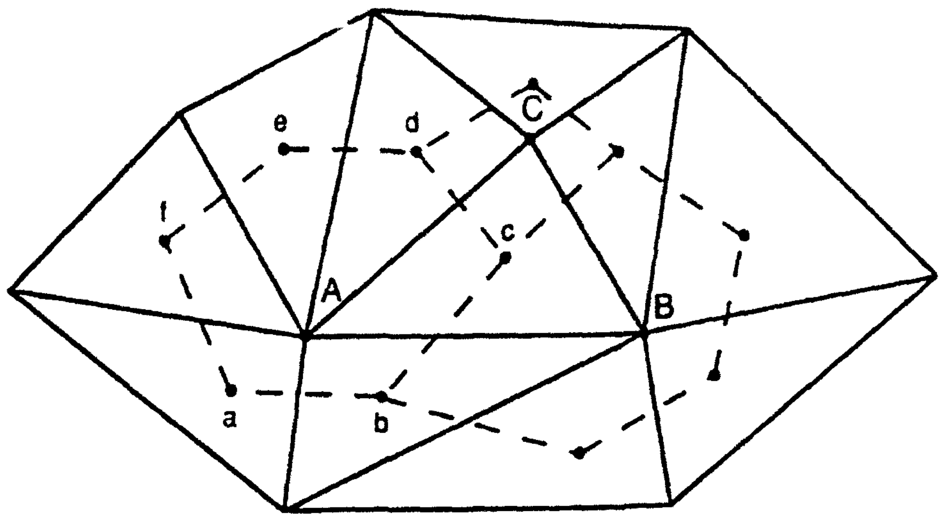
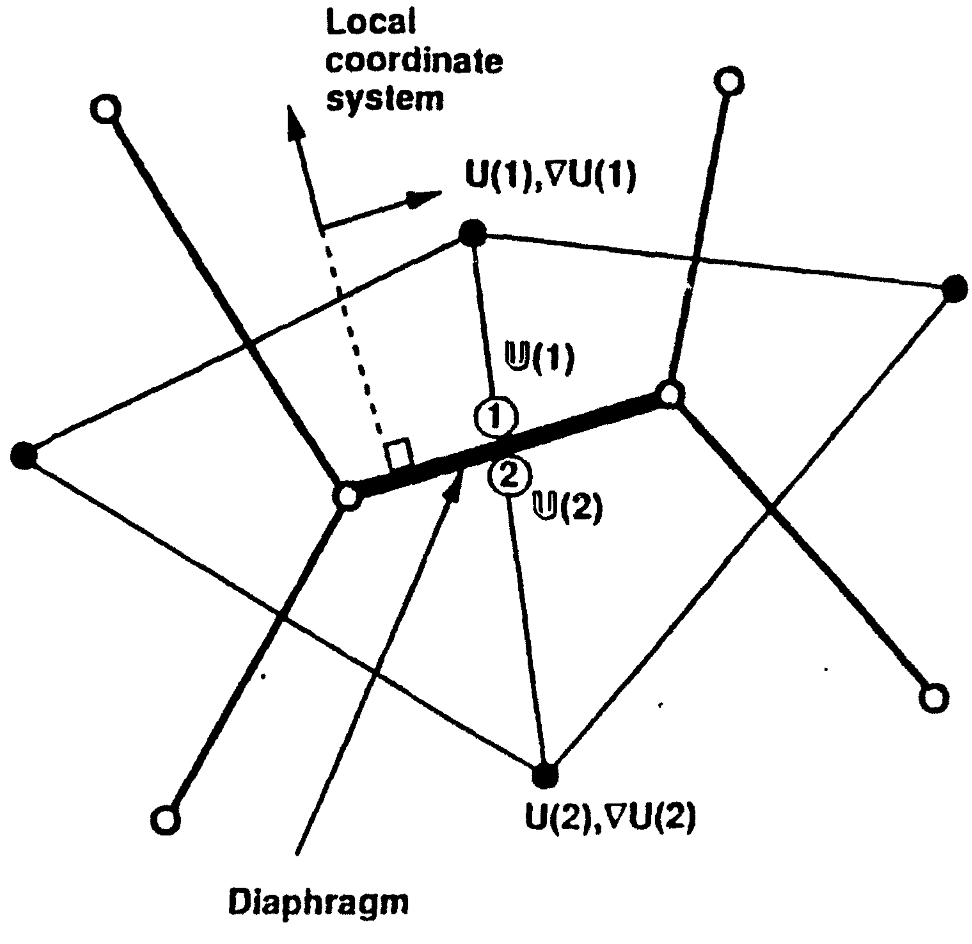
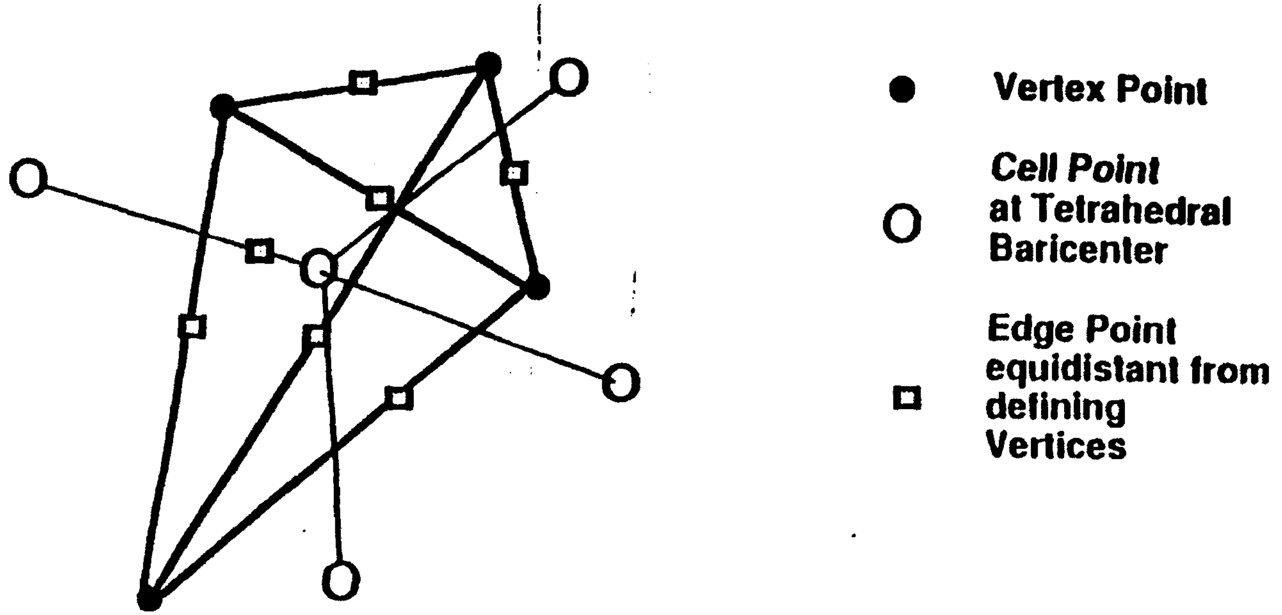


Figure 4.



Second Order Edge Based Flux Calculation

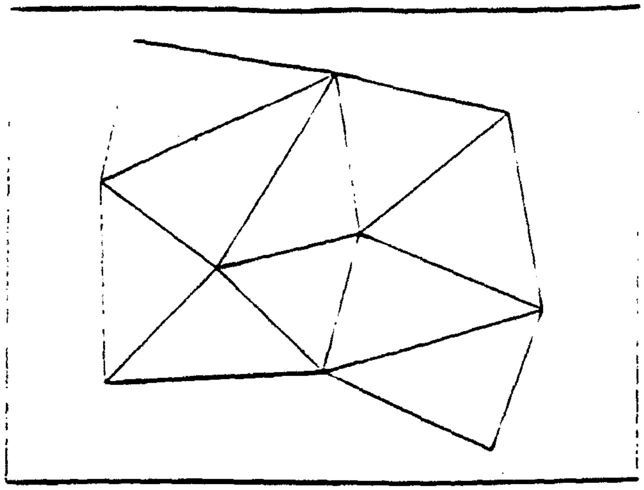
Figure 5.



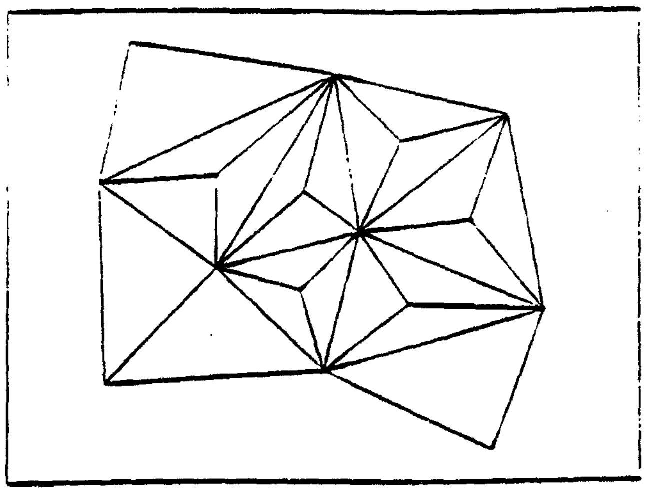
Tetrahedral Element
Defined by four Vertices
and Baricentric Cell Point

Scheme for Baricentric Three Dimensional Integration

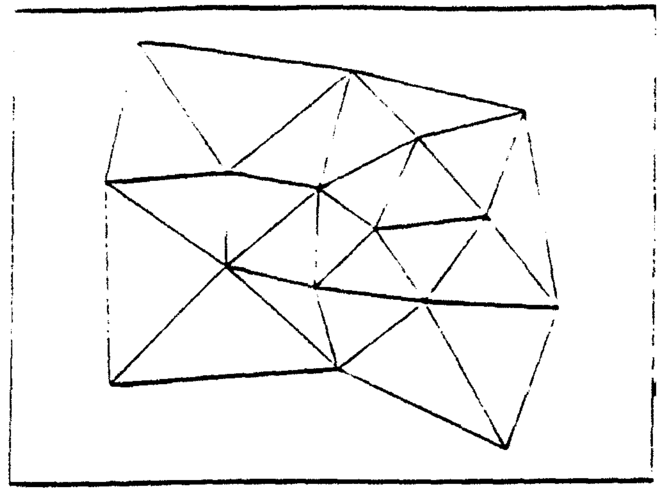
Figure 6.



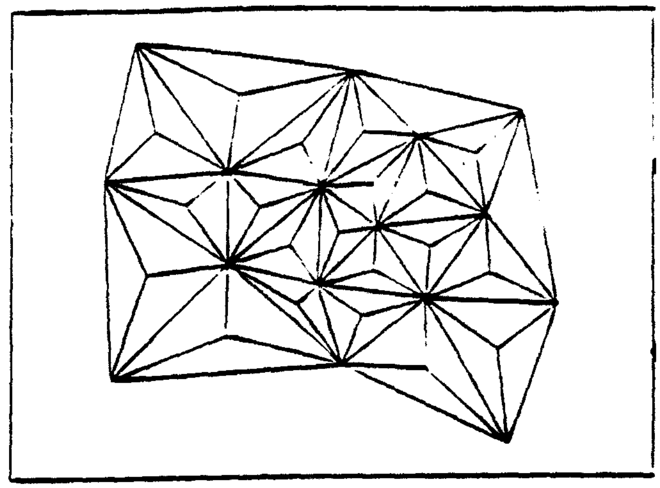
a. Original grid.



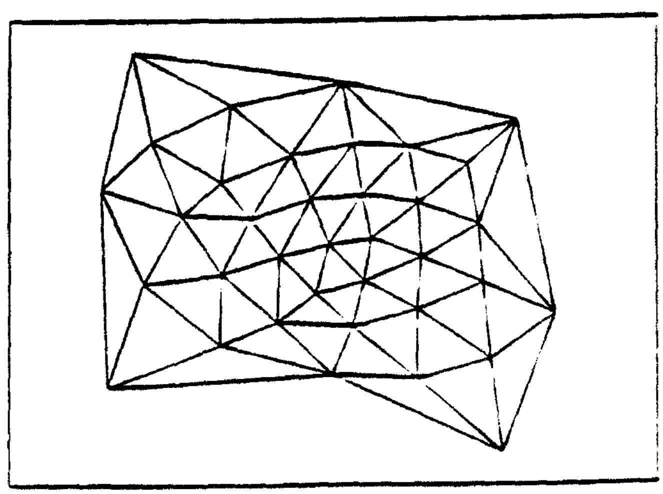
b. Grid after one refinement.



c. Grid after one refinement and one reconnection.

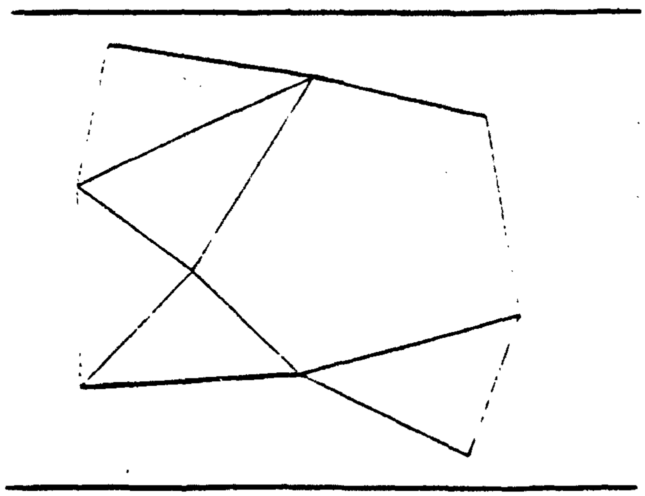


d. Second refinement.

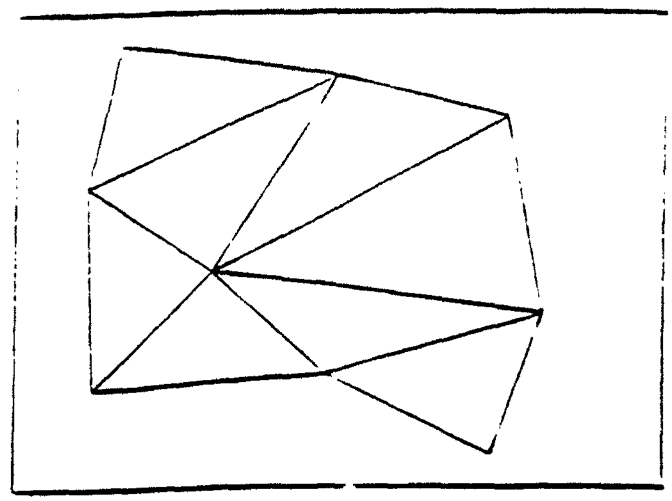


e. Second reconnection.

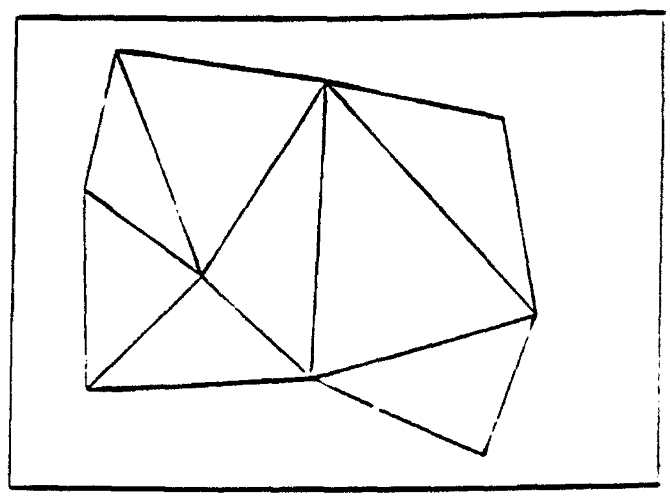
Figure 7. Illustration of the grid refinement process.



a. Point removal.

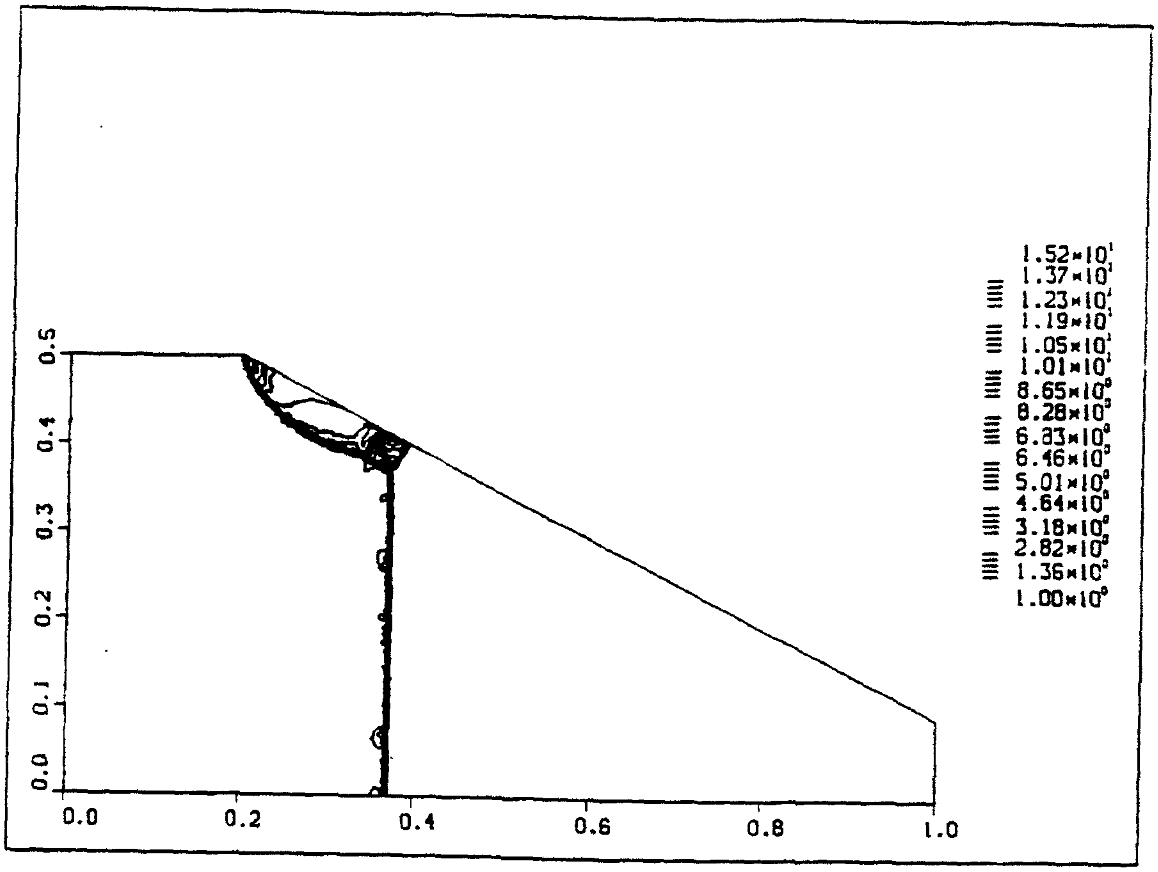


b. Construction of new cells.

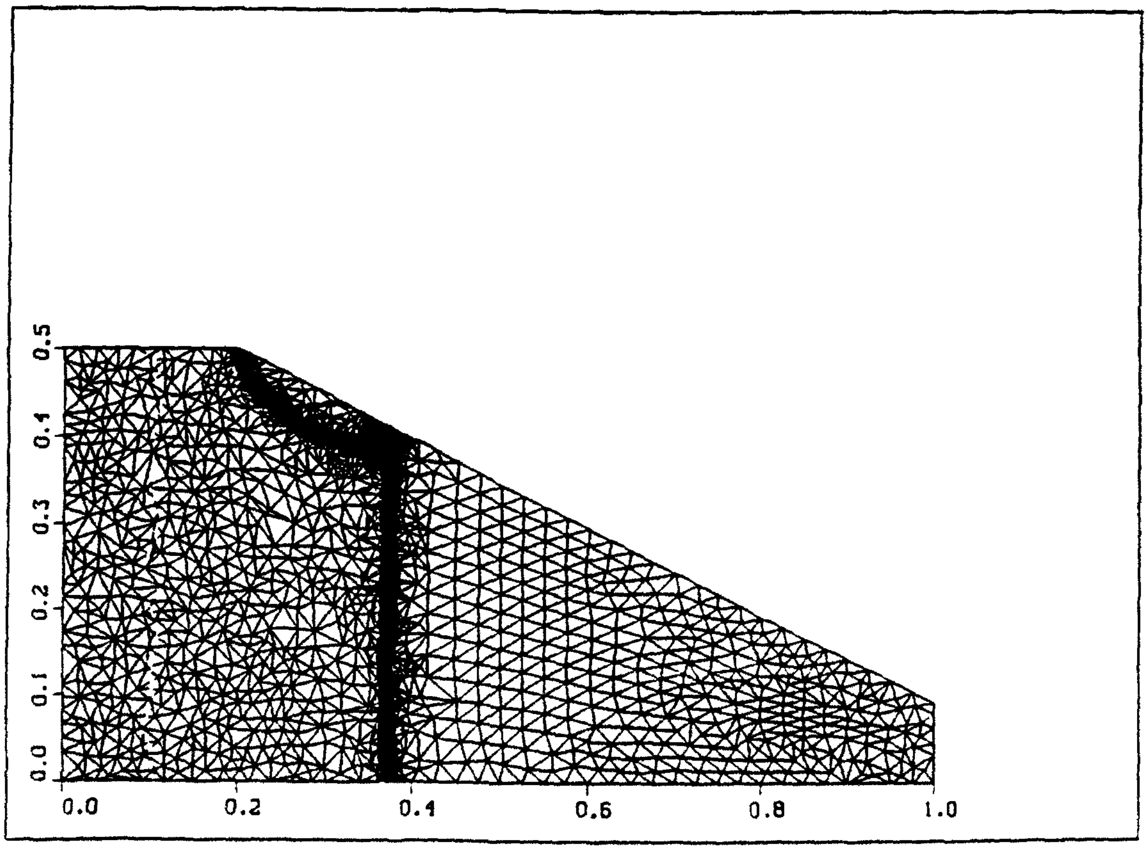


c. Final coarse grid after reconnection.

Figure 8. Illustration of the grid coarsening process.

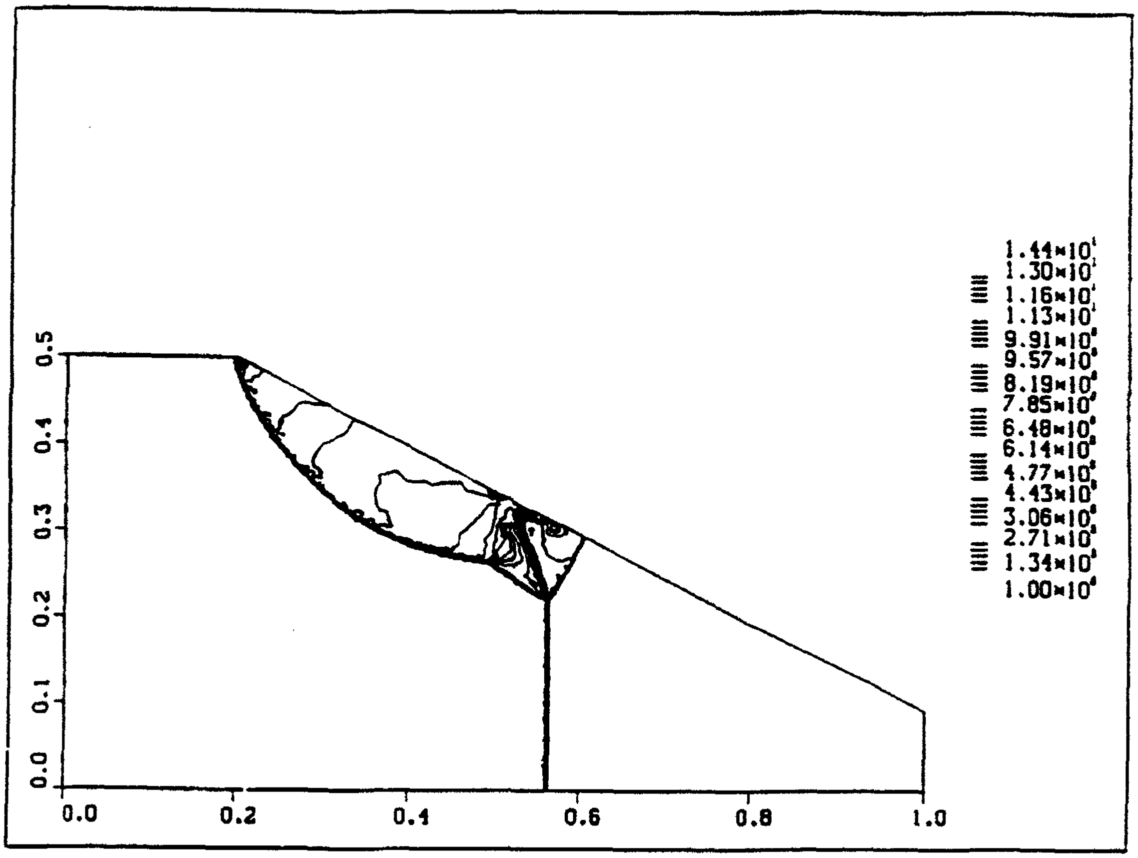


Density contours

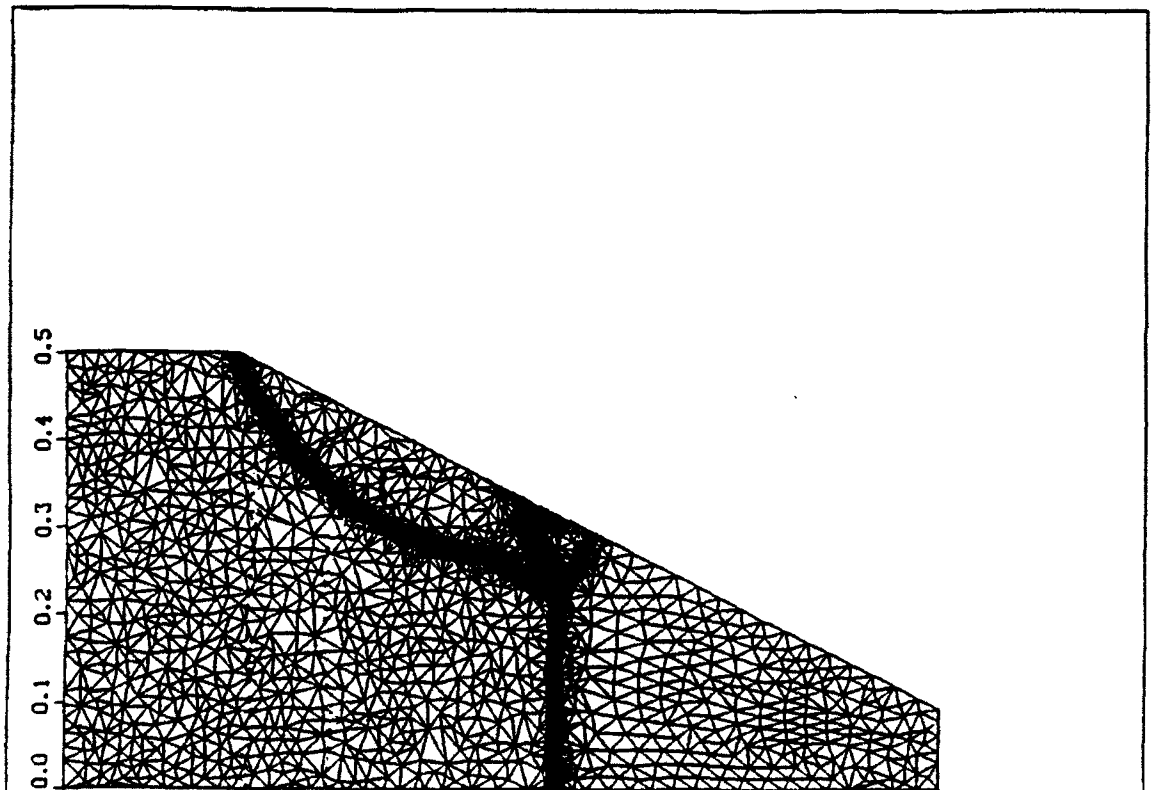


Grid

Figure 8a. Mach Reflection in a planar channel. $M_s = 8.7$; $\alpha = 27°$.

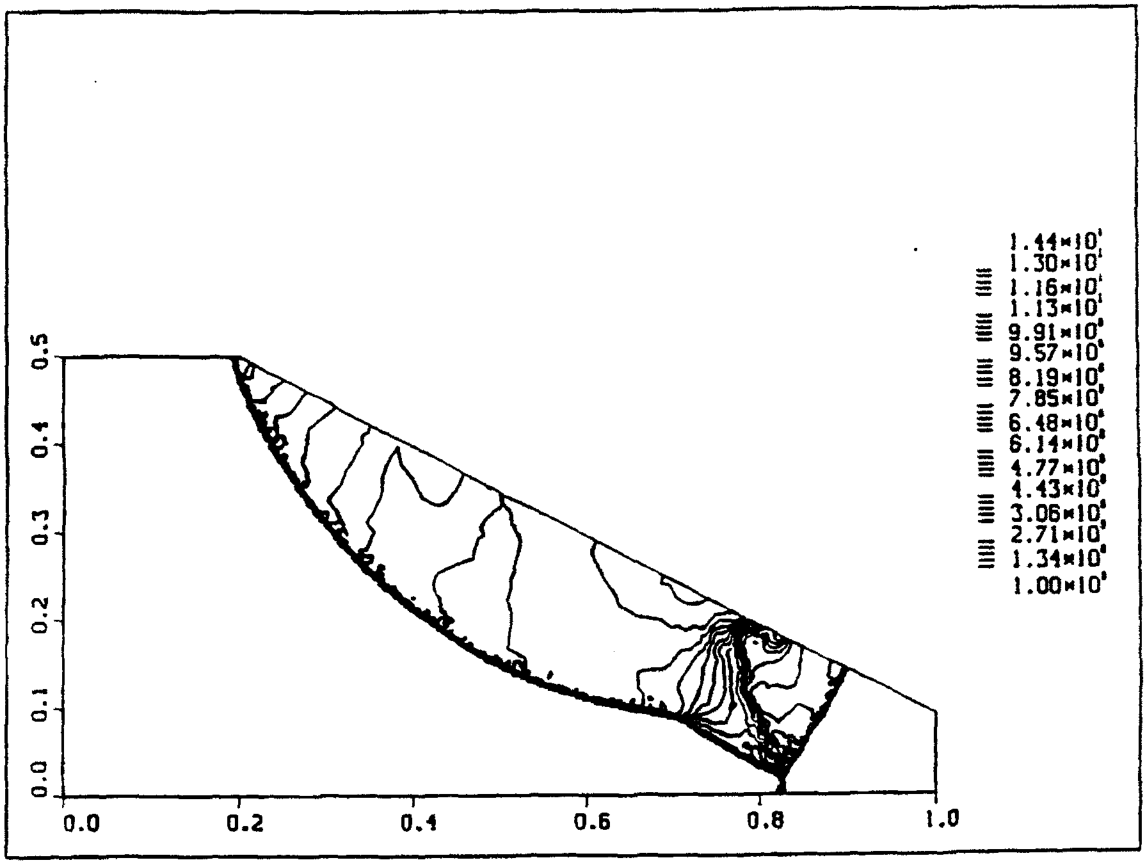


Density contours

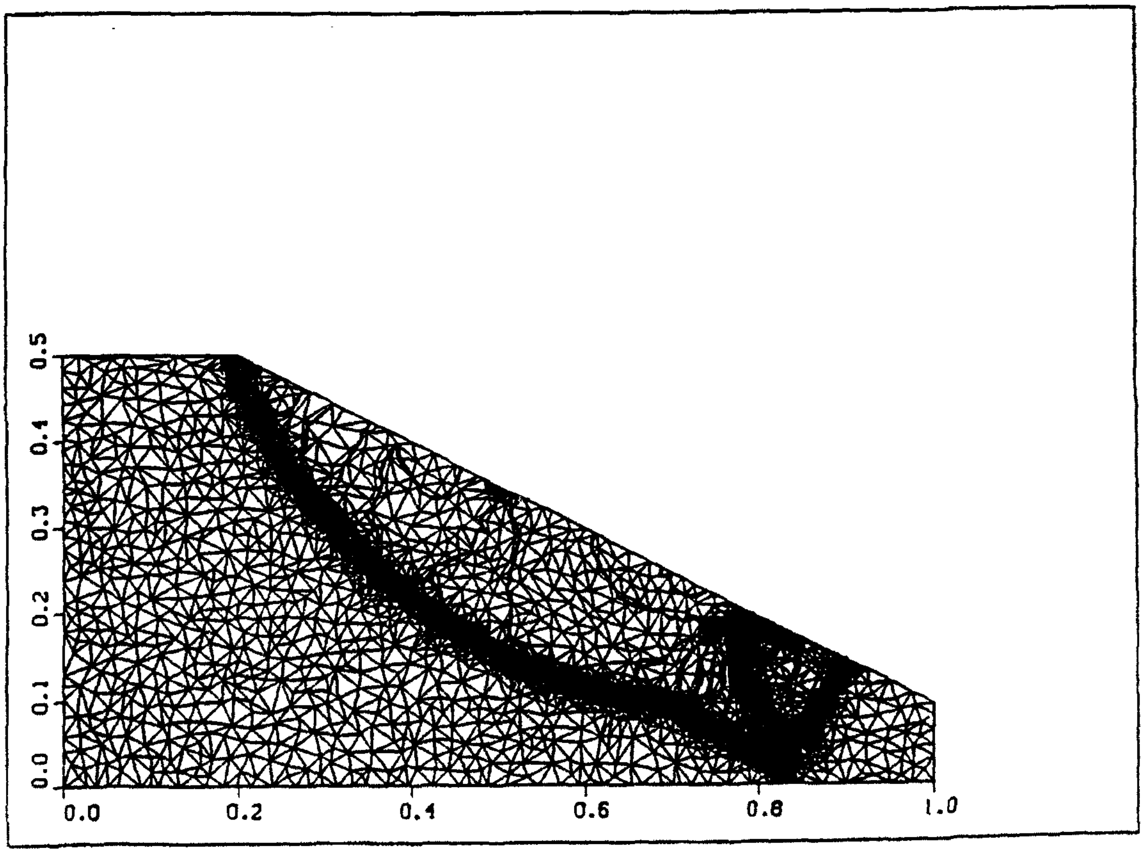


Grid

Figure 8b. Mach Reflection in a planar channel. $M_s = 8.7$; $\alpha = 27°$.



Density contours



Grid

Figure 8c. Mach Reflection in a planar channel. $M_s = 8.7$; $\alpha = 27°$.

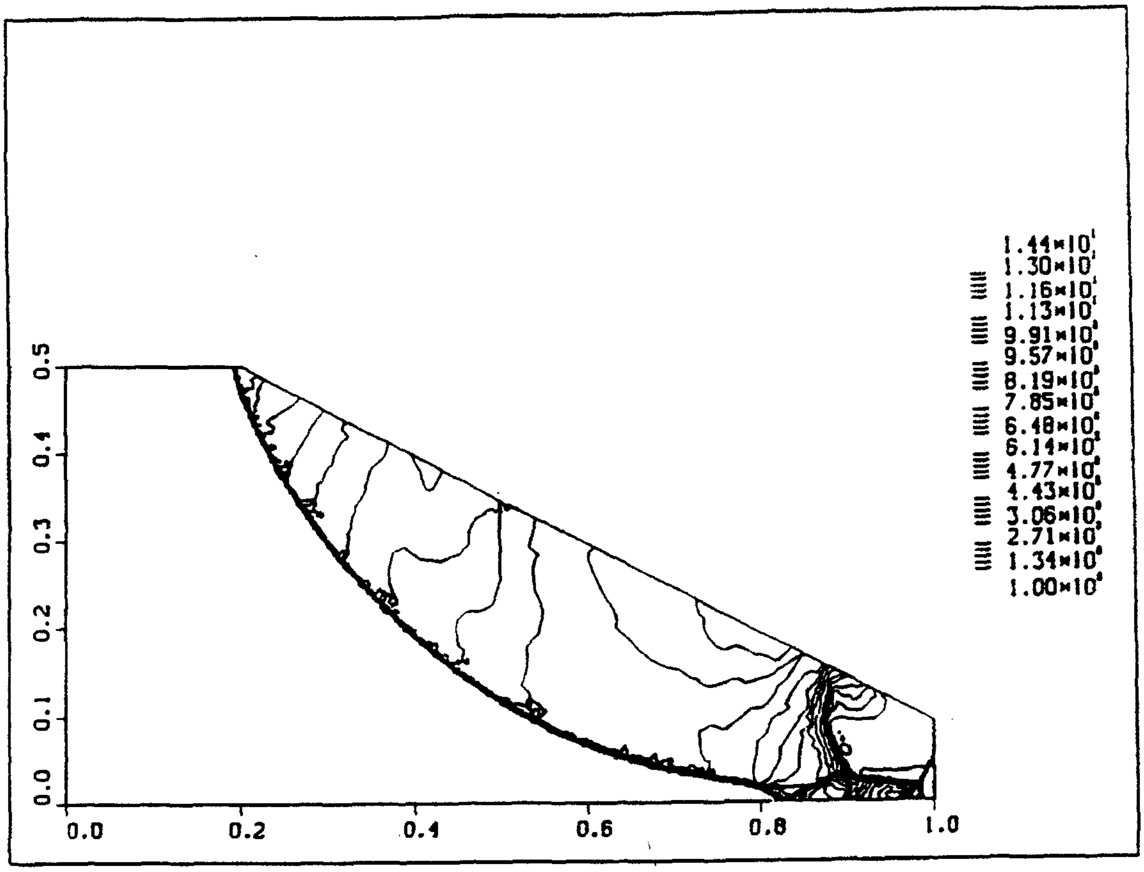
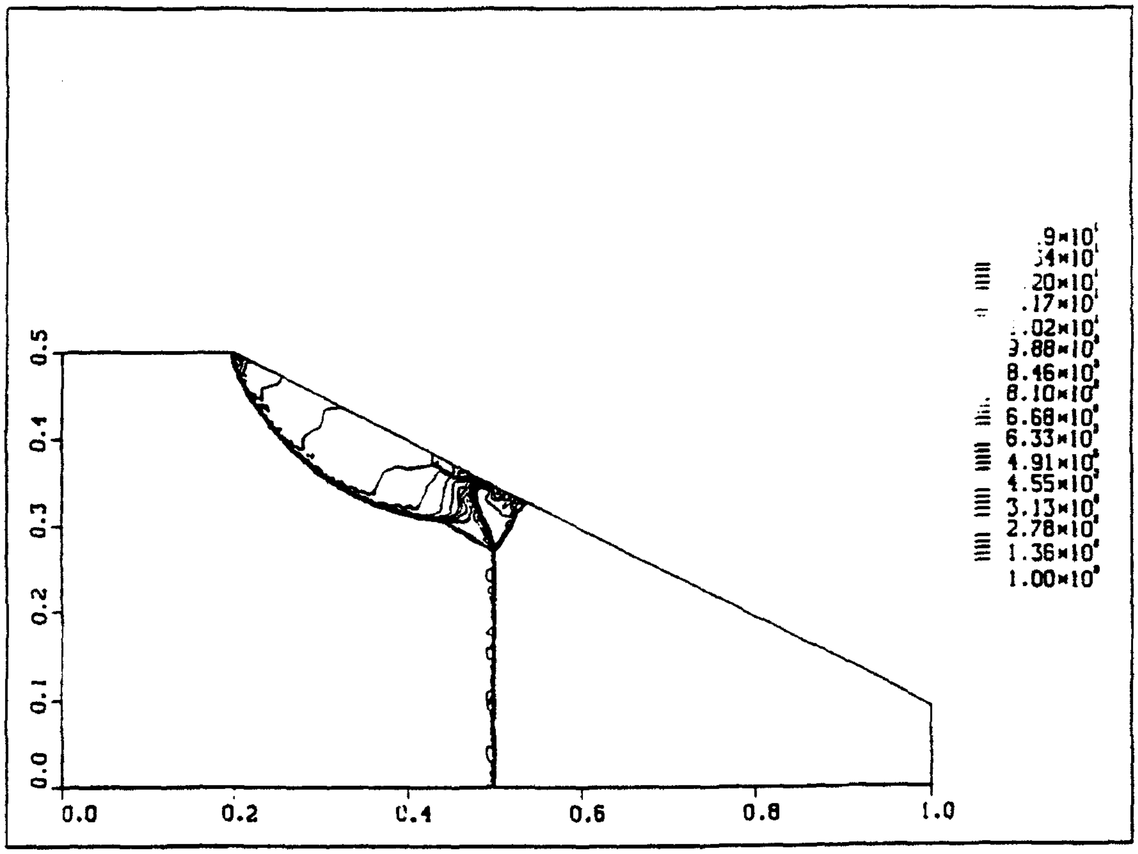
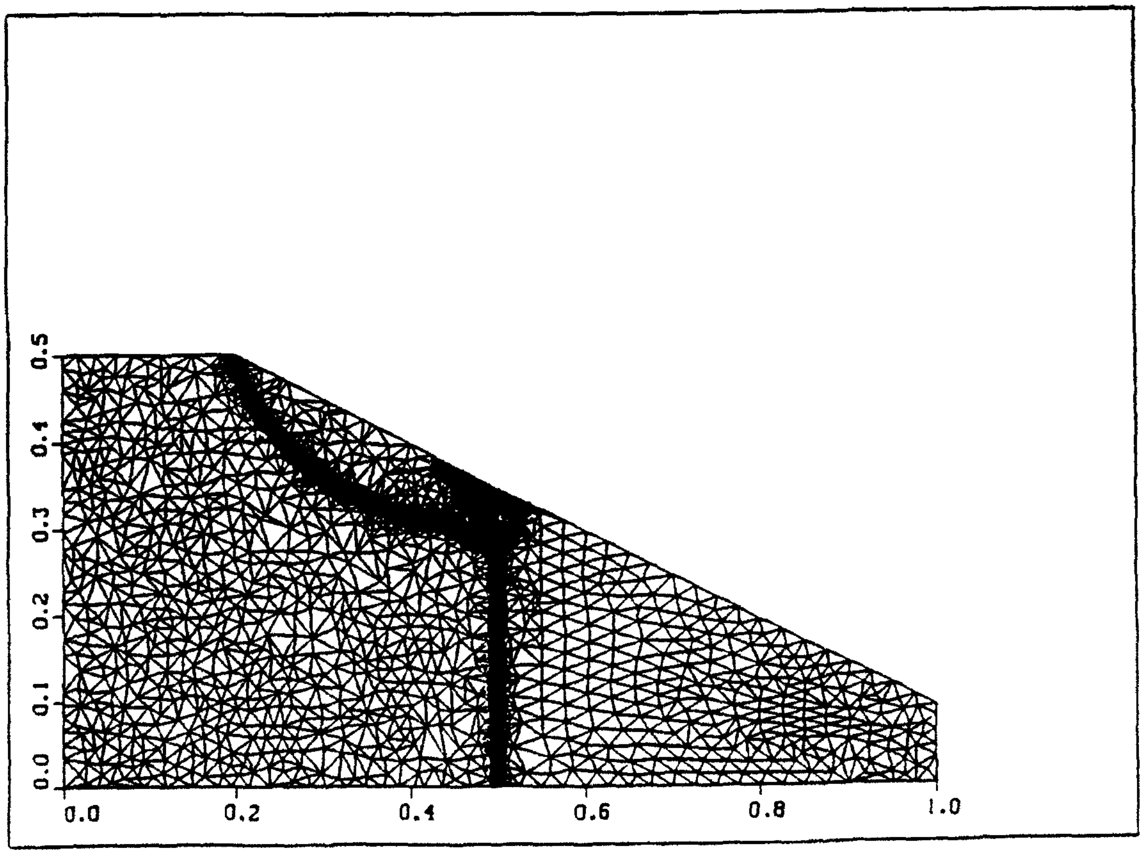


Figure 9a. Secondary Mach Reflection in a planar channel. Density contours.

Figure 9b. A blown up view of the secondary Mach Reflection shown in Figure 9a.

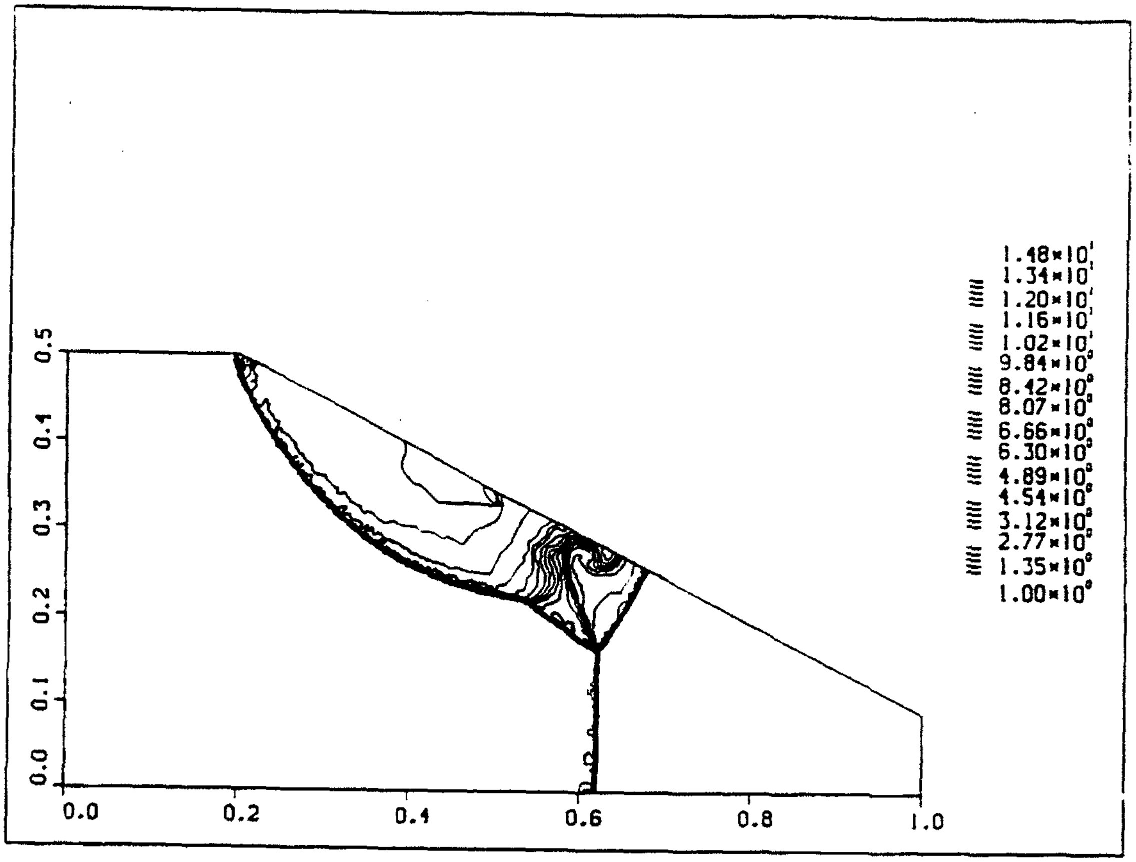


Density contours

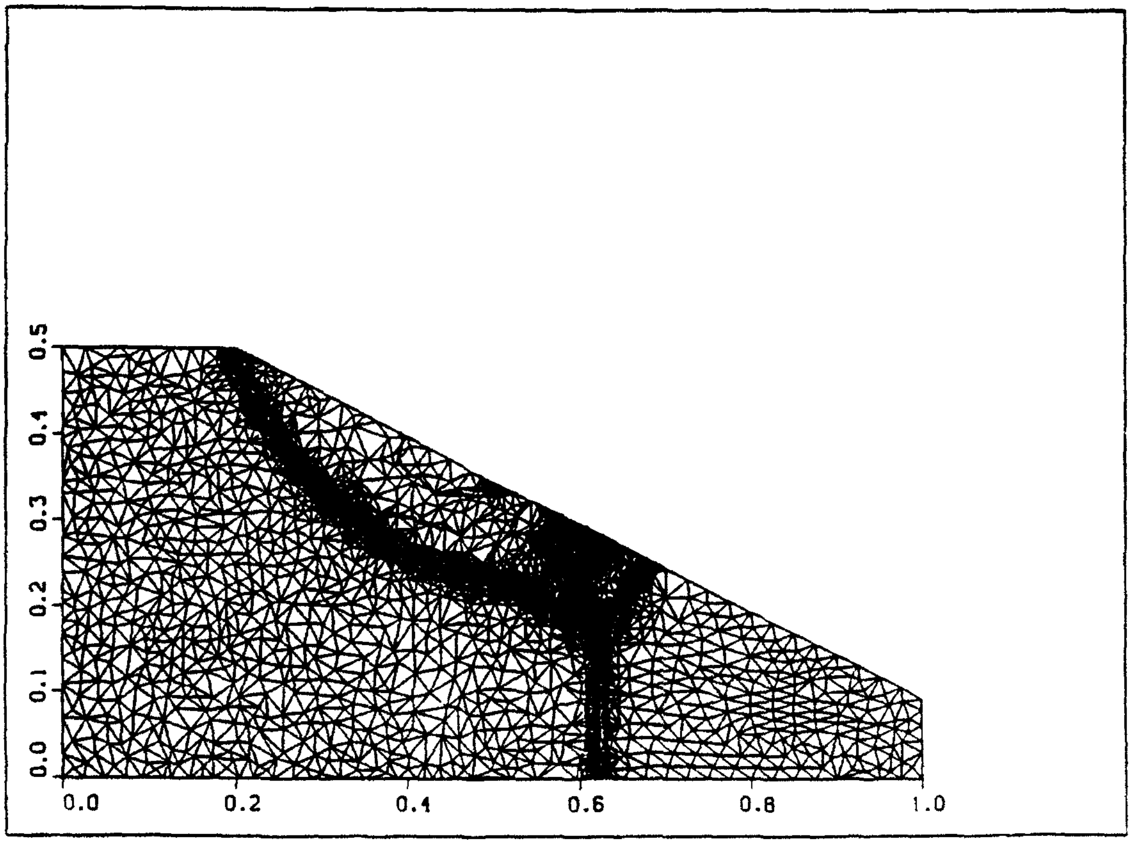


Grid

Figure 10a. Mach Reflection in an axisymmetric channel. $M_s = 8.7$; $\alpha = 27°$.

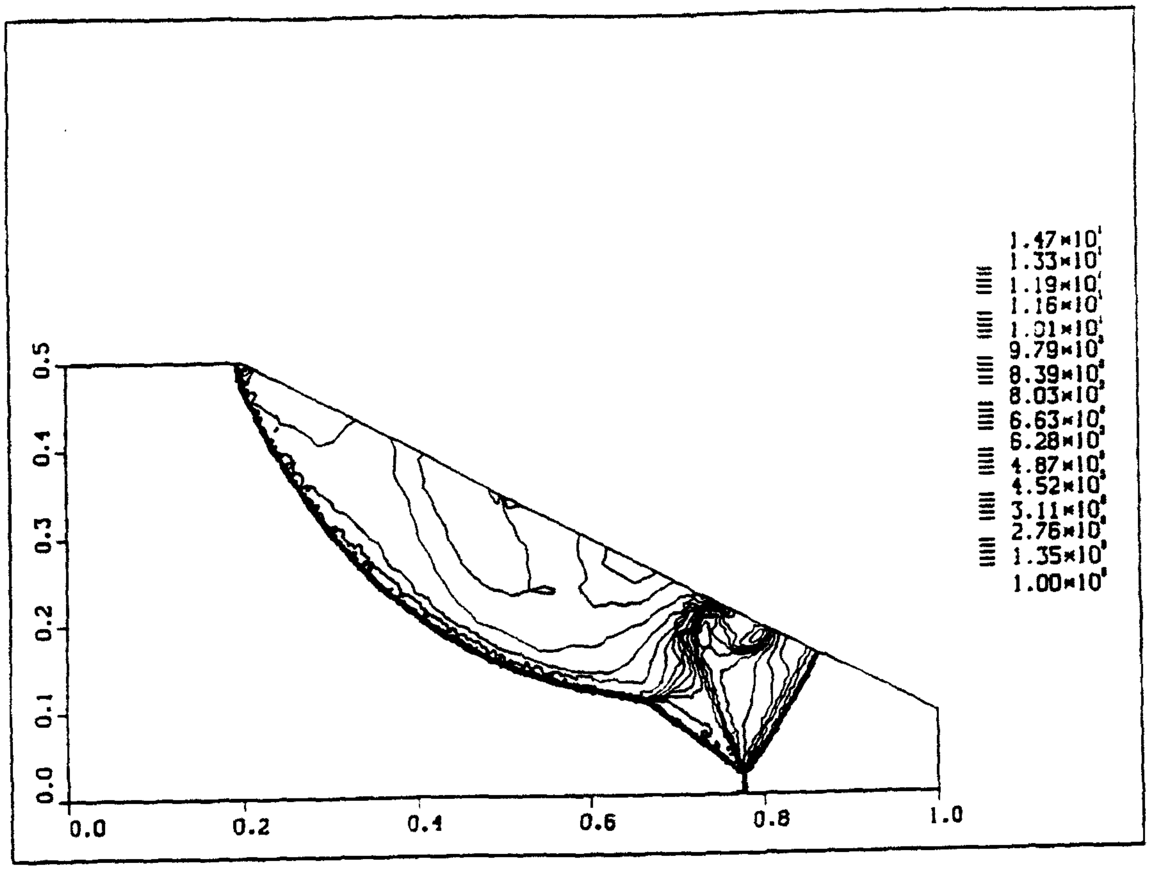


Density contours

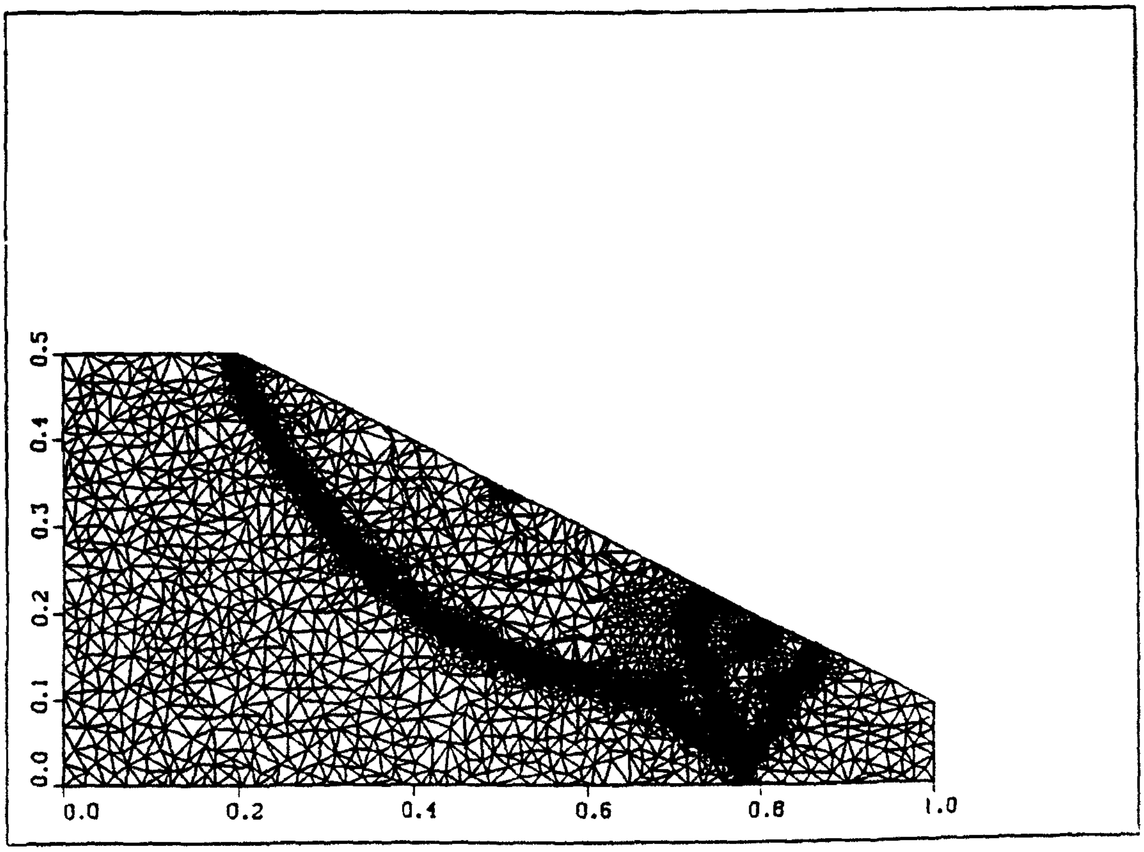


Grid

Figure 10b. Mach Reflection in an axisymmetric channel. $M_s = 8.7$; $\alpha = 27°$.



Density contours



Grid

Figure 10c. Mach Reflection in an axisymmetric channel. $M_s = 8.7$; $\alpha = 27°$.

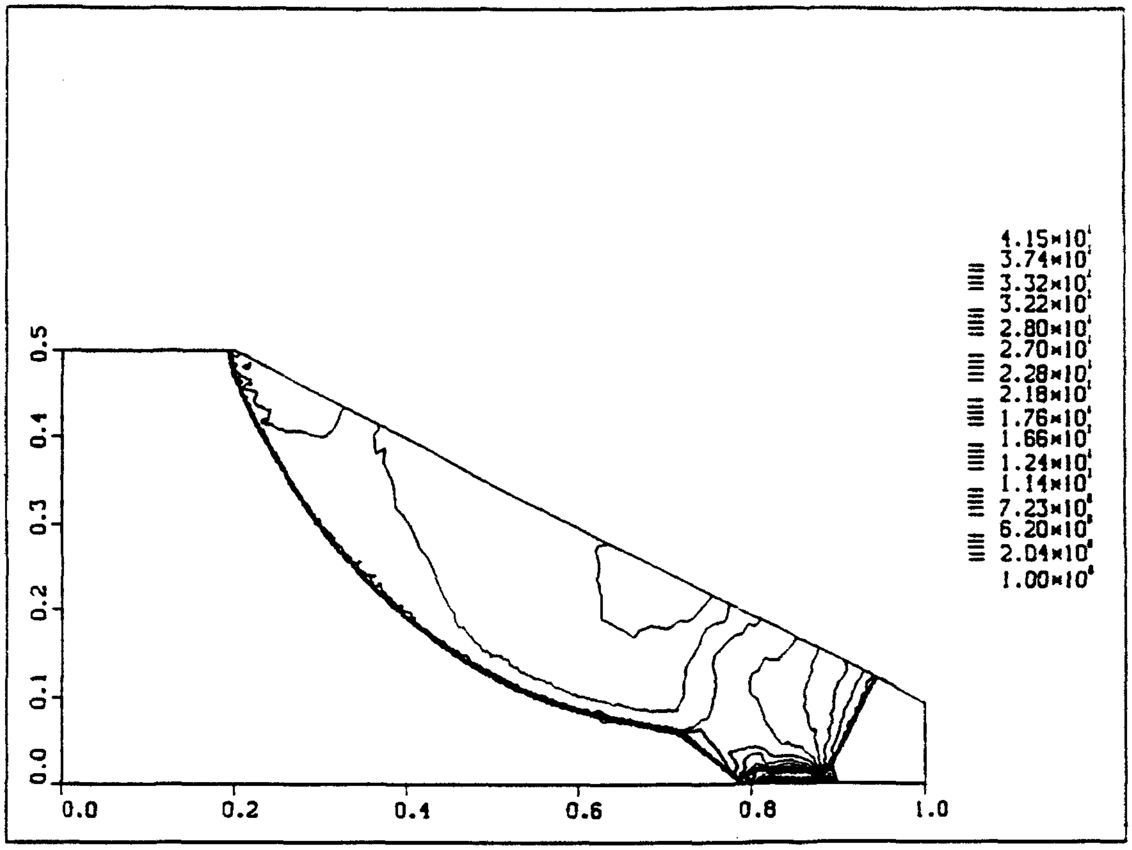


Figure 11a. Start of the secondary Mach Reflection. Axisymmetric channel. Density contours.

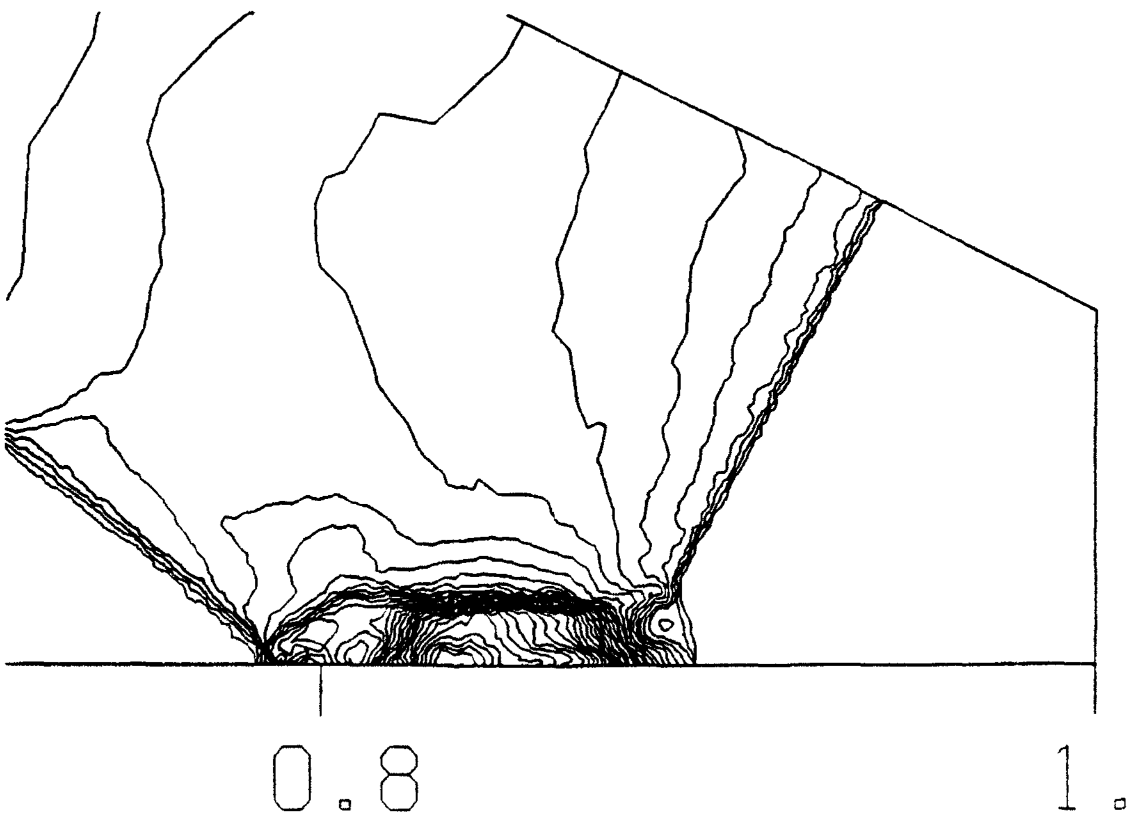


Figure 11b. Blow up of the secondary Mach Reflection area shown in Figure 11a.

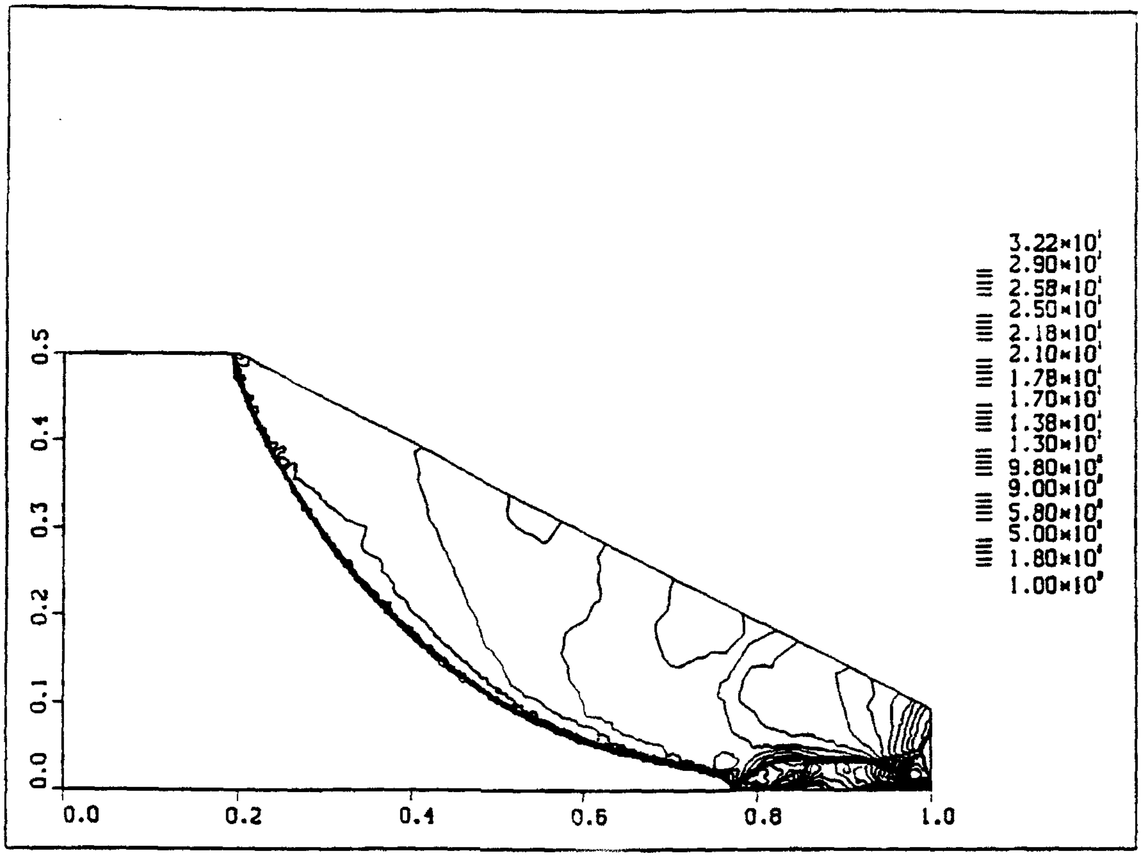


Figure 11c. Secondary Mach Reflection axisymmetric channel. Density contours.

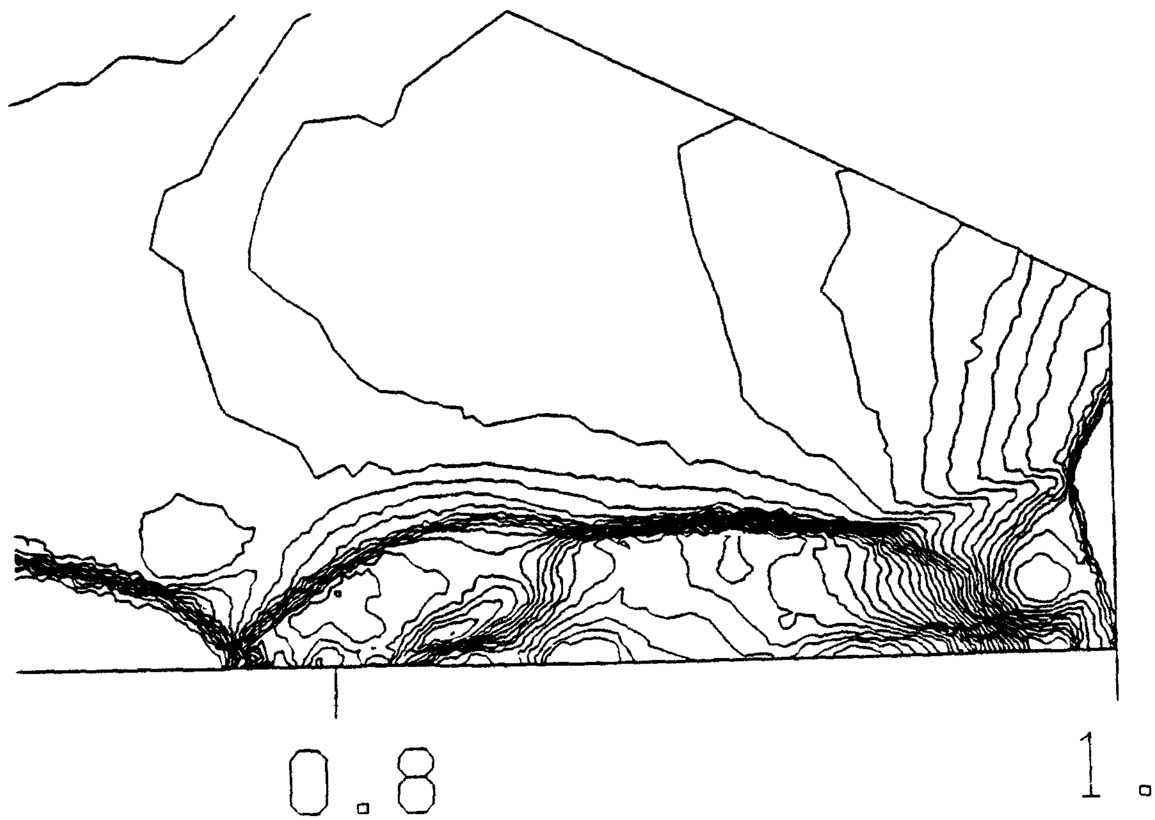


Figure 11d. Blow up of the secondary Mach Reflection area shown in Figure 11c.

Solution of Euler's Equations on Adaptive Grids Using A Fast Unstructured Grid Second Order Godunov Solver (FUGGS)

Itzhak Lottati, Shmuel Eidelman and Adam Drobot
Science Applications International Corporation
1710 Goodridge Drive
McLean, Virginia 22102

Abstract. We describe a new technique for solving Euler's gasdynamic equations on unstructured triangular grids with arbitrary connectivity. The formulation is based on the second order Godunov method. The use of data structure with only one level of indirectness leads to an easily vectorized and parallelized code with a low level of overhead in memory requirement and high computational efficiency. The performance and accuracy of the algorithm has been tested for a very wide range of Mach numbers starting from very low subsonic to high hypersonic flows, without the need to adjust any code parameters. The algorithm was implemented in a vertex based and triangle based scheme. The computational results produced by the triangle based version showed an extremely low level of artificial viscosity.

A new method of direct dynamic refinement of unstructured grids, as described in this paper, allows an automatic adaptation of the grid to regions of pressure or density discontinuity, steep pressure or density gradient, and high vortical activity. Results using the algorithm with dynamic grid refinement are presented.

Flow Solver on an Unstructured Grid

The specific use of triangular meshes provides a very flexible means for simulating flows in extremely complex geometries. The data that identifies a triangular mesh (unstructured grid) provides the flexibility needed to properly discretize the complex geometry of the computational domain, especially on the boundary where the geometry and the implementation of boundary conditions, are extremely crucial for the accuracy of the simulation. The flexibility of unstructured grids enables adaptation to physical features in the flow. The price of resolution results in a local rather than a global penalty. Consequently, it is possible to simulate problems on computers with limited memory and still achieve highly resolved solutions. A typical example, which is illustrated in this paper, is a travelling shock passing over an obstacle. The challenge is to simulate such problems with fine resolution across the shock while limiting the total number of mesh points in the calculation.

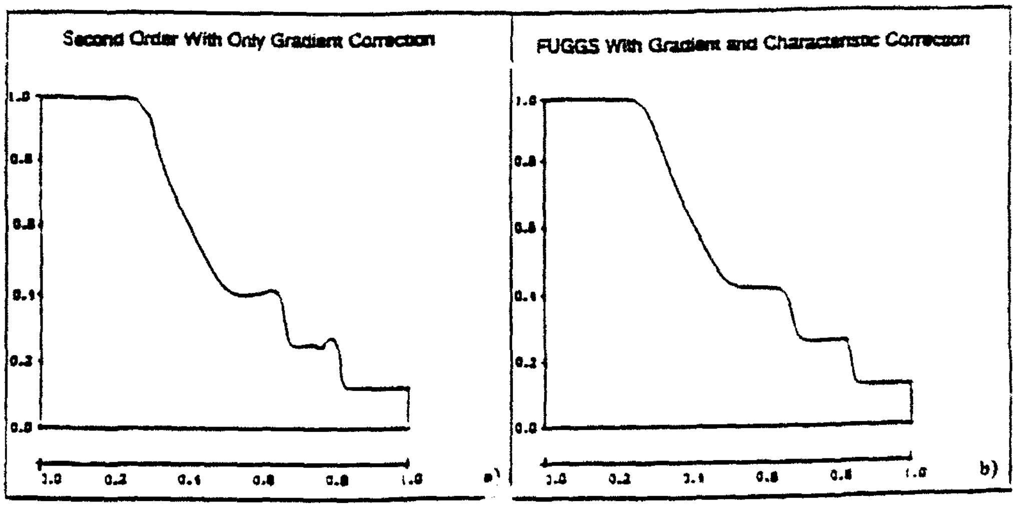


Figure 2: Sod problem, effect of the characteristics correction on the density.

Performance and Validation of FUGGS

FUGGS has proven to be a very robust algorithm capable of high quality solutions while using triangles with large variations of aspect ratios. The code was tested on a variety of unstructured grids and consistently provided results, despite the apparent poor quality of the underlying mesh. We were able to simulate efficiently and accurately a wide spectrum of flow regimes starting from low subsonic to high hypersonic. The code has no free parameters to choose and thus does not require any "tuning" to specific problems. The user has only to specify the boundary conditions (around the grid) and initial flow conditions. The algorithm is fully vectorized and can be easily parallelized in the future. We describe the detailed algorithm below and then present typical results.

Direct Dynamic Refinement Method for Unstructured Triangular Grids

As stated, an unstructured grid is very suitable to implement boundary conditions on complex geometrical shapes and refinement of the grid if necessary. This feature of the unstructured triangular grid is compatible with efficient usage of memory resources. The adaptive grid enables the code to capture moving shocks and high gradient flow features with high resolution. The memory resources available can be very efficiently distributed in the computational domain to accommodate the resolution needed to capture the main features of the physical property of the solution. Dynamic refinement controls the resolution

of the grid according to available memory resources and subject to prescribed priorities. These priorities can be set according to the physical features which the user wishes to emphasize in the simulation. The user has control over the resolution of the physical features resolved in the simulation, without being restricted to the initial grid. The alternative to Direct Dynamic Refinement is the hierarchical dynamic refinement (H refinement) that keeps a history of the initial grid (mother grid) and the subdivision of each level (daughters grid). The H refinement subdivides the initial grid into two or four triangles in each level, and keeps track of the number of subdivision levels each triangle has undertaken. In the H refinement method, one has to keep overhead information on the level of each triangle subdivision, and needs double indirect indexing to keep track of the H refinmenet process. This slows down the computation by partially disabling the vectorization of the code. As mentioned, the H refinement does rely heavily on the initial grid as it subdivides the mother grid and returns back to it after the passage of the shock.

Direct Dynamic Refinement for capturing the shocks basically requires the refinement to be in the region ahead of the shock. This requirement minimizes the dissipation in the interpolation process when assigning values to the new triangles created in the refined region. Additionally, it requires that the coarsening of the grid should be done after the passage of the shock. In principle, the interpolation and extrapolation in the refinement and coarsening of the grid is done in the region where the flow features are smooth.

The physics of the problem should be involved in the process that identifies the region of refinement and coarsening. One can derive error criteria that will allow grid adaptation to stationary or moving pressure or density discontinuities, region of high voritical activity, etc. For each of the physics features to be resolved, there should be an error indicator that is suited best to capture and identify the region of importance corresponding to this feature.

Criteria for Refinement (Error Indicator)

We have implemented an algorithm with multiple criteria for capturing a variety of features in the physics of the problem to be solved. That means that we were able to derive a number of error indicators that enable identification of moving shock waves or stationary shocks in the computational domain.

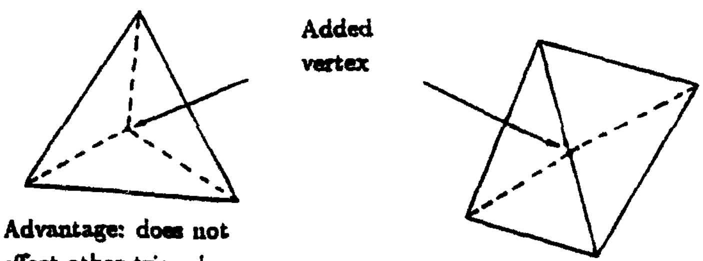
To identify the location of a moving shock, we use the flux of energy or momentum into triangles. The fluxes entering and leaving triangles are the most accurate physical variables computed by the Godunov algorithm for solving the Euler's equations, and are used to update the physical variables for each time step in each triangle. A shock wave means that there is a "step function" change in the cell that is caused by an influx of energy, momentum or density.

Stationary shock can be identified by density gradients computed as required in the second order Godunov algorithm.

The refinement process is done in two ways: i) adding a vertex in the center of a triangle and ii) adding a vertex on an edge of a triangle. Figure 3 illustrates the two alternative ways used to refine the grid. Figure 4 shows an example of the refinement procedure. In the coarsening stage we identify a vertex to be removed. With the point removal, we delete the connecting edges and triangles surrounding the point. The next step is to triangulate the void polygon by creating new triangles using only the vertices of the polygon. Figure 5 shows an example of how the coarsening proceeds.

In the process of refinement and coarsening, we often create triangles with large aspect ratios (the base-to-height maximum ratio for the three edges). We use reconnection to flip the diagonal between two adjacent triangles to obtain triangles with a "better" aspect ratio. This procedure is referred to as the reconnection step in Figs.4 and 5.

- Adding a vertex in barycenter of triangle.
- Adding a vertex on the middle of an edge of a triangle.

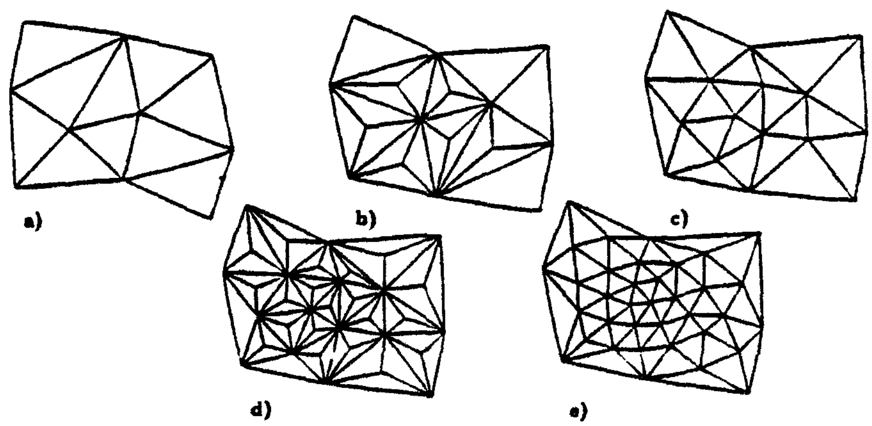


Advantage: does not effect other triangles.

Disadvantage: effects the aspect ratio of the triangles.

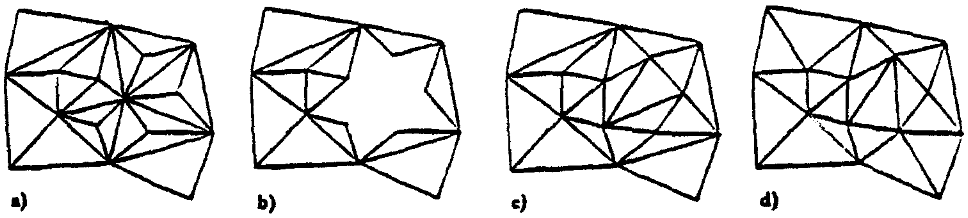
This method is used on the boundary to improve the triangles with acute angles.

Figure 3: Two Ways of Refinement.



a) Original grid. b) One refinement. c) First reconnection.
d) Second refinement. e) Second reconnection.

Figure 4: Illustration of the grid refinement process.



a) Original grid. b) Point removal. c) Constructing of new cells.
d) Coarse grid after reconnection.

Figure 5: Illustration of the grid coarsenning process.

Results

Direct dynamic refinement was used to solve the transient behavior of the flow entering a channel with a double wedge having an inclination of 20°. The flow Mach number entering the channel is 2.5. The flow is from left to right. A sequence of snapshots illustrates the density contours, and the grid for each timestep is given in Figs. 6 (countour plots) and 7 (grid). These figures clearly demonstrate the automatic adaptation of the grid to the moving shocks and the ability to capture the detailed physics of the simulation with very high resolution and minimal memory requirements. The initial grid can clearly be seen to the right of the shock ("ahead") in the early stage of the shock propagation from left to right. The coarsening algorithm is able to produce a reasonable mesh in the region trailing the shock as shown in Fig. 7.

The ability to capture stationary shocks is illustrated in Fig. 8 in which a supersonic free flow ($M = 2.5$) has been run over a diamond shape bump (20° wedge) driven to a steady state. The shock emerges from the first corner (left), the fan of rarefaction waves appears from the apex of the diamond shape bump, and the secondary shock from the second corner (right) is clearly illustrated by the ability of the algorithm to adapt the grid to the physics of the flow. Figure 9 illustrates the sharpness of the reflected shock obtained for an axisymmetric converged channel with an angle of 27° and $M = 8.7$.

The few examples shown here represent a small subset of results obtained with FUGGS. The examples are indicative of the excellent performance that can be achieved for physically complicated situations. We would like to emphasize that these calculations involved no free parameters.

Acknowledgment

We would like to thank Dr. Aharon Friedman and Dr. Marty Fritts for their valuable contributions. This work was partially supported by the Wright Research and Development Center under contract No. F33615-88-C-3002 and by DARPA through contract F49620-89-C-0087 administered by AFOSR.

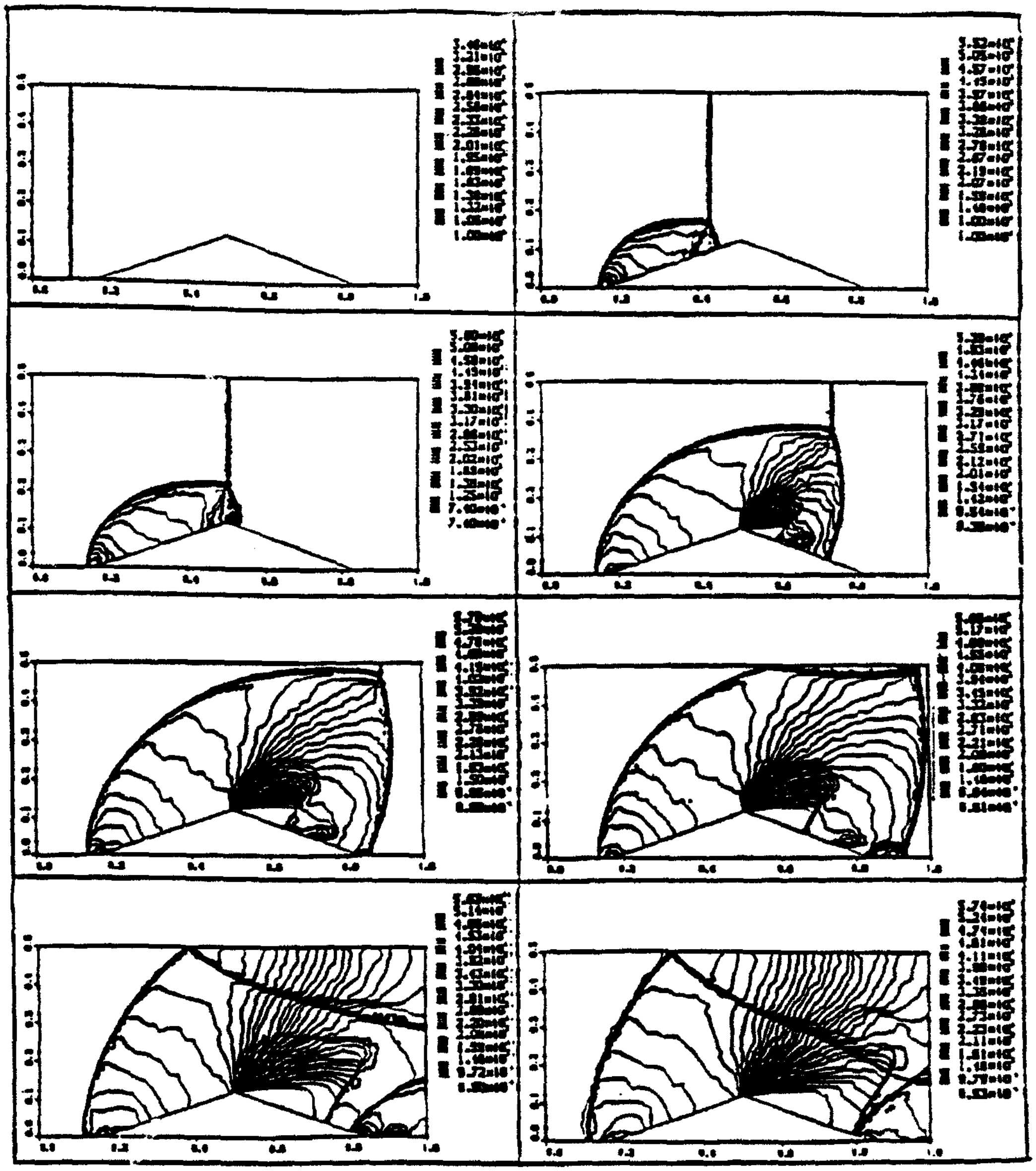


Figure 6. A sequence of density snapshots of countour plots for a propagating shock ($M = 2.5$, wedge angle $= 20°$).

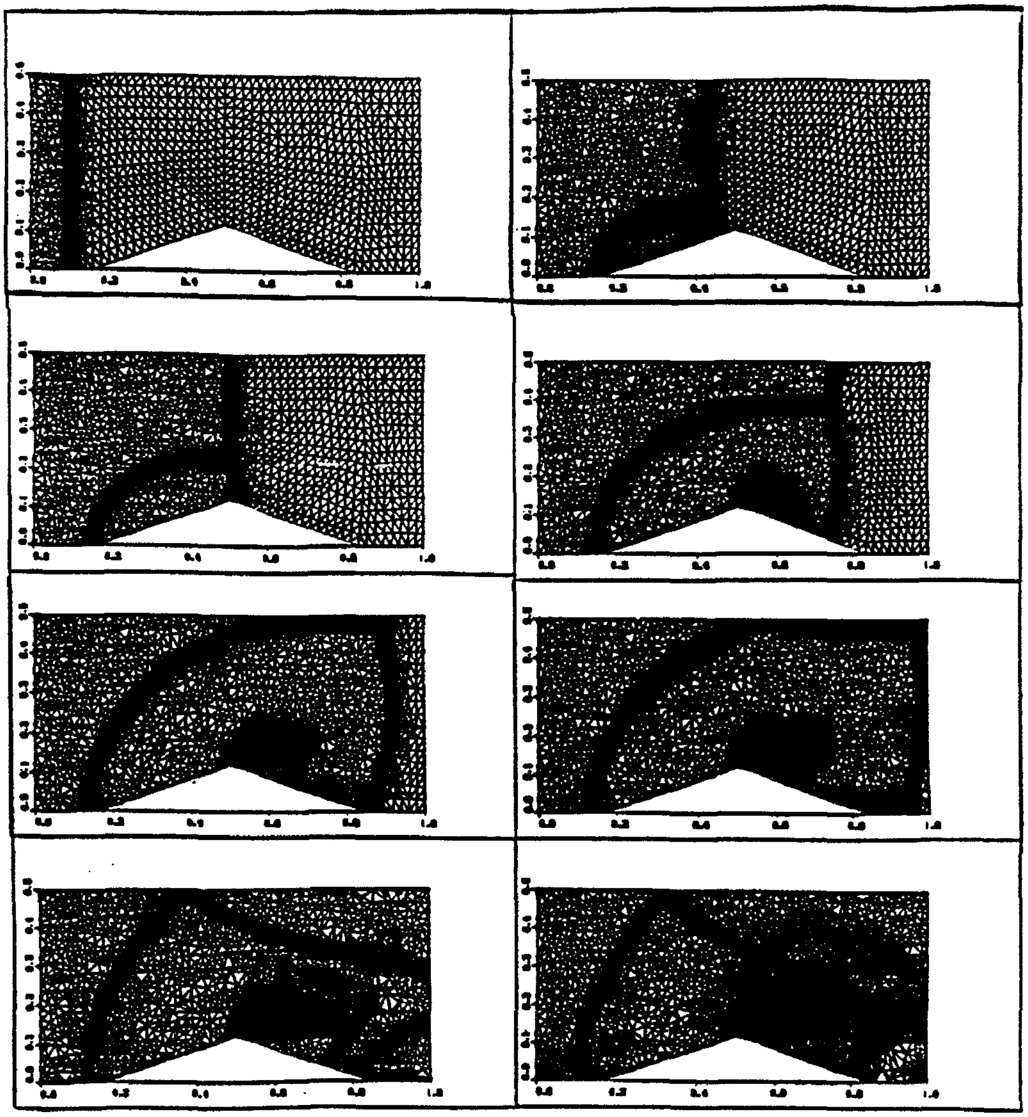


Figure 7. A sequence of grids corresponding to countour plots in figure 6.

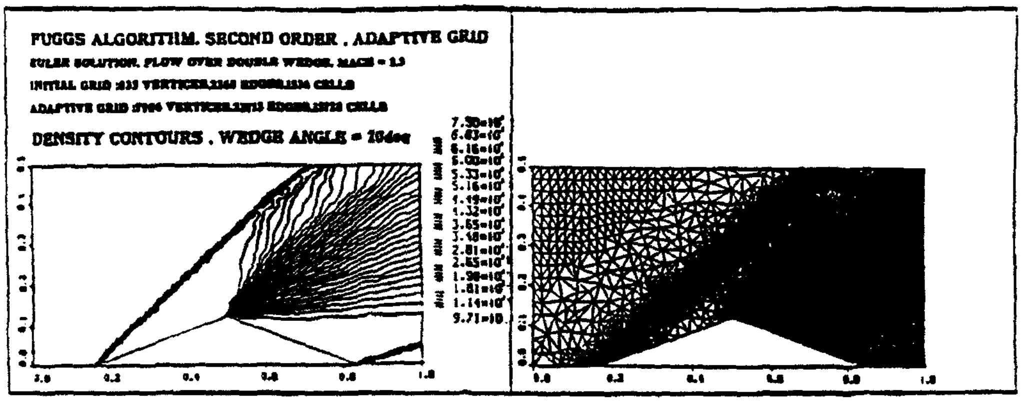


Figure 8. Density contour plot and grid for flow driven to steady state over a double wedge obstacle.

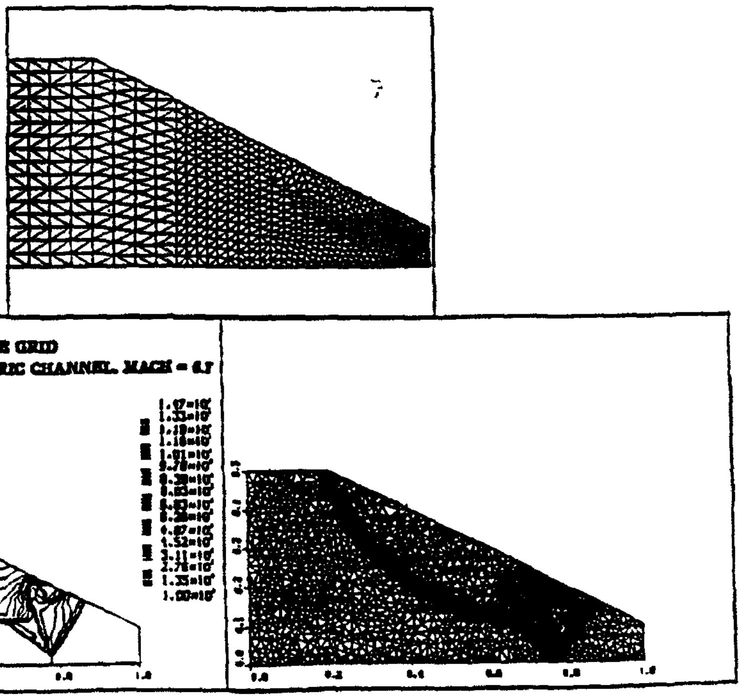


Figure 9. Initial grid, countour plot and the adaptive grid for flow in axisymmetric channel (M = 8.7, wedge angle = 27°).

AIAA

AIAA-90-2420

Air-Breathing Pulsed Detonation Engine Concept; A Numerical Study

S. Eidelman, W. Grossmann and I. Lottati
Science Applications International Corporation,
McLean, Va

AIAA/SAE/ASME/ASEE
26th Joint Propulsion Conference
July 16-18, 1990 / Orlando, FL

AIR-BREATHING PULSED DETONATION ENGINE CONCEPT: A NUMERICAL STUDY

S. Eidelman, W. Grossmann and I. Lottati

Applied Physics Operation
Science Applications International Corporation
1710 Goodridge Drive
McLean, Virginia 22102

1. Introduction

The airbreathing Pulsed Detonation Engine (PDE) concept was introduced by us and reported on in the past[1,2,3]. As described in the previous reports, we have carried out a systematic series of parametric studies of the PDE via Computational Fluid Dynamics (CFD) and have analyzed engine performance over a wide range of flight regimes including subsonic and supersonic flows and physical geometries including various nozzle and air inlets. In addition, we have performed static table top experiments[1] to demonstrate that the principle of pulsed or repetitive detonation can be achieved in a generic PDE configuration. To date, our results indicate that practical engines for certain vehicles and missions can be conceptualized and designed with the information that has already been generated from the studies. Specifically, our studies have shown that the PDE is an excellent candidate for the primary propulsion source for small aerodynamic vehicles that operate over the flight envelope, $0.2<M<3$ and altitude between sea level and 30,000 ft. Further, our analysis of the simulation results indicate that the PDE is a high thrust to weight ratio device with a specific fuel consumption on the order of one pound per hour per pound fuel. The predicted performance places the PDE propulsion concept in a strongly competitive position compared with present day small turbojets. The PDE concept has the added attractiveness of rapid variable thrust control, no moving parts and the potential for low cost manufacturing. Finally, the PDE concept is scalable over a wide range of engine sizes and thrust levels. For example, it is theoretically possible to produce PDE engines on the order of one to several inches in diameter and thrusts on the order of pounds, as well as devices which provide thousands of pounds thrust.

The parametric studies that we have carried out to date were possible due to the development of a new generation of CFD tools that have allowed us to accurately simulate the details of the complex nonlinear time dependent processes. A brief description of the CFD methods employed in our studies is given in section 3.

The purpose of the present paper is: (1) to report the most recent studies of a full simulation of the operation of the PDE with a generic missile configuration cruising at supersonic speeds, (2) to report the results of a parametric-scaling study of the thrust produced as a function of the variation of a given engine configuration with respect to engine size.

The present paper is organized as follows: Section 2 gives, for completeness, a brief description of the PDE concept, Section 3 describes briefly the CFD methods used in our most recent studies, Section 4 gives the results of the parametric-scaling study and, Section 5 describes the simulations of the complete flow around a generic missile configuration powered by a PDE, Section 6 gives our summary and conclusions.

2. The Pulsed Detonation Engine Concept

A detonation process, due to the very high rate of reaction, leads to a propulsion concept in which the constant volume process can be fully realized. In detonative combustion, the strong shock wave, which is part of the detonation wave, acts like a valve between the detonation products and fresh charge. The speed of the detonation wave is about two orders of magnitude higher than the speed of a typical deflagration. This allows the design of propulsion engines with a very high power density. Each detonation has to be initiated separately by a fully controlled ignition device, with a wide range of variable cycle frequencies. There is no theoretical restriction on the range of operating frequencies; they are uncoupled from acoustical chamber resonance. This is very important feature of the constant volume detonation process that differentiates it from the process occurring in a pulse-jet;[4,5] the pulse jet cycle is tuned to the acoustical resonances of the combustion chamber. This leads to a lack of scalability for the pulse jet concept.

A physical restriction dictating the range of detonation frequency arises from the rate at which the fuel/air mixture can be introduced into the detonation chamber. This also means that a device based on a detonative combustion cycle can be scaled and its operating parameters can be modified for a range of required output conditions. There have been numerous attempts to take

advantage of detonative combustion for engine applications. The most recent and successful of these attempts was carried out at the Naval Postgraduate School (NPS) by Helman et al.[1] During this study, several fundamentally new elements were introduced to the concept distinguishing the NPS research device from previous studies. First, it is important to note that the NPS experimental apparatus was the first successful self aspirating air breathing detonation device. Intermittent detonation frequencies of 25 Hz were obtained. This frequency was in phase with the fuel mixture injection through timed fuel valve opening and spark ignition. The feasibility of intermittent injection was established. Pressure measurements showed conclusively that a detonation process occurred at the frequency chosen for fuel injection. Further, self aspiration was shown to be effective. Finally, the effectiveness of a primary detonation as a driver for the main detonation was clearly demonstrated. Although the NPS studies were abbreviated, many of the technical issues considered to be essential for efficient intermittent detonation propulsion were addressed with positive results.

The generic device we consider here is a small engine shown in Figure 1. Figure 1 shows a schematic of the basic detonation chamber attached to the aft end of a generic aerodynamic vehicle. The combustible gas mixture is injected at the closed end of the detonation chamber and a detonation wave propagates through the mixture. The size of the engine suggests a small payload or aerodynamic vehicle, but the concept can be extended to larger payloads simply by scaling up the size of the detonation chamber and possibly combining a number of chambers into one larger engine.

A key issue in the pulsed detonation engine concept is the design of the main detonation chamber. The detonation chamber geometry determines the propulsion efficiency and the duration of the cycle (frequency of detonations). Since the fresh charge for the generic engine is supplied from the external flow field, the efficiency of the engine depends on the interaction of the surrounding flow with the internal flow dynamics. The range of the physical process requiring simulation in order to model the complex flow phenomena associated with the detonation engine performance is very broad. A partial list is:

1. Initiation and propagation of the detonation wave inside the chamber,
2. Expansion of the detonation products from the chamber into the air stream around the chamber at flight Mach numbers,
3. Fresh air intake from the surrounding air into the chamber.
4. The flow pattern inside the chamber during post-exhaust pressure buildup which determines the strategy for mixing the next detonation charge,
5. Strong mutual interaction between the flow inside the chamber and surrounding the engine.

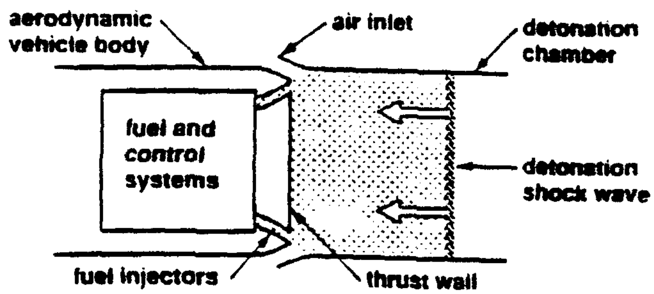


Figure 1. Schematic of the generic PDE showing detonation chamber, inlet, detonation wave, fuel injectors and position relative to an aerodynamic vehicle.

All of these processes are interdependent, and interaction and timing are crucial to engine efficiency. Thus, unlike simulations of steady state engines, the phenomena described above can not be evaluated independently. The need to resolve the flow regime inside the chamber accounting for nozzles, air inlets etc., and at the same time resolve the flow outside and surrounding the engine, where the flow regime varies from high subsonic, locally transonic and supersonic, makes it a challenging computational problem.

The single most important issue is to determine the timing of the air intake for the fresh charge leading to repetitive detonations. It is sufficient to assume inviscid flow for the purpose of simulating the expansion of the detonation products and fresh air intake. The assumption of inviscid flow makes the task of numerically simulating the PDE flow phenomena somewhat easier than if a fully viscous flow model were employed. For the size of the generic device studied in this work the effects of viscous boundary layers are negligible with the exception of possible boundary layer effects on the valve and inlet geometries discussed subsequently..

3. *Computational* Methods used in the Studies

The basic computational tool that was used for our studies is the FUGGS (Fast Unstructured Grid Second Order Godunov Solver) code, described in detail in Refs. 6,7. This code provides a method for solving the Euler equations of gasdynamics on unstructured grids with arbitrary connectivity. The formulation is based on a second order Godunov method[8]. The use of a data structure with only one level of indirectness leads to an easily

vectorized and parallelized code with a low level of overhead in memory requirement and high computational efficiency. The performance and accuracy of the algorithm has been tested for a very wide range of Mach numbers and geometrical situations, and has demonstrated robustness without the need for any adjustable parameters. The algorithm can either be triangle or vertex based; experience with the method has shown that extremely low levels of artificial viscosity can be achieved using the triangle based version of the method.

A new method of direct dynamic refinement of unstructured grids has been developed, (Ref. 6), and allows an automatic adaptation of the grid to the region of the moving detonation wave inside the PDE geometry. This refinement guarantees that the associated highly inhomogeneous pressure and density contours of the detonation wave are accurately tracked in the simulation. This is an important ingredient in our simulations, since the main component of the detonation process contributing to the thrust generated by the PDE is the total kinetic energy of the wave. Use of the new refinement scheme has more accurately describe the moving detonation wave behavior. These new results concern nonplanar wave evolution and, as pointed out in Section 4, may be a factor in controlling the magnitude of the generated thrust.

4. Scaling Study of the PDE

We have shown in our previous study that in the Pulsed Detonation Engines, thrust is primarily produced by the unsteady interaction of shock wave generated by the propagating detonation wave and the thrust wall of the detonation chamber. This interaction will be nonlinear and scalability of the engine will greatly depend on the extent of nonlinearity. For example, for the engine geometry shown in Figure 1, the engine volume can be increased just by elongating the wall of the detonation chamber. If the area of the thrust wall in Figure 1 remains the same and the composition of the detonation mixture does not change, the increase in the detonation chamber length will result in longer duration of the interaction between the shock wave and the thrust wall. This simple situation poses a question concerning the relationship between the increase in PDE thrust and increase in its volume. This is very practical issue in scaling up the size of the engine, since increase in the detonation chamber diameter will eventually result in difficulty generating a planar detonation front, leading to loss of engine efficiency.

To study this aspect of the detonation engine scalability we have conducted a set of numerical simulations for the engine geometry very similar to these shown in Figure 1. The detonation chamber diameter was kept constant at 8cm and its length varied from 8cm to 16cm. The main objective of our study is to determine how the thrust produced by the detonation engine increases when the engine length doubles and the rest of the engine parameters will remain the same. This section describes the results of two simulations for the detonation chamber geometry described above, using a detonation chamber length of 8 cm and 16 cm. The simulation begins at $t = 0$ when the detonation chamber is placed in an external freestream with the Mach number of 0.8. The detonation wave is initiated at the aft end of the detonation chamber. The detonation chamber for these cases includes a simple annular inlet which remains open during operation. The specific fuel chosen for the present simulations is ethylene. The chemical reaction occurring in the ethylene/air detonation process is given by:

$$C_2H_4 + 3O_2 + 11.24N_2 \longrightarrow 2H_2O + 2CO_2 + 11.24N_2.$$

The detonability limits of ethylene in air range from 4% to 12% by volume and depend somewhat on temperature and pressure. We assume for the sake of simplicity that the fuel/air ratio is 6% by volume. In contrast with our previous presentations here, as well as in case of supersonic PDE simulation presented in this paper, we have simulated a propagating detonation wave by releasing the energy of detonative combustion in our mixture immediately behind the detonation front. In our simulations we have used the Dynamically Adaptive FUGGS code which we have developed recently. Figures 2a, 2b, and 2c, present three frames of the results for simulation in a 16 cm long detonation chamber. In these figures, results are presented in the form of pressure contour plots. For illustration of the dynamic grid adaptation to the evolving flow pattern, we have plotted the unstructured triangular grid corresponding to the stage at which contour plots are shown. In Figure 2a, pressure contour plots are shown shortly after the detonation wave has been initiated at the aft end of the detonation chamber. We can observe that the shock wave front is planar. The detonation wave velocity is 1800 m/sec and the pressure at the front of the detonation wave is $\approx$ 20 atm., corresponding to the CJ condition for the ethylene/air mixture. Figure 2b shows the results of the detonation wave reflecting from the thrust wall and the detonation products starting to expand into the flow stream surrounding the detonation chamber. The detonation products expand through the inlet and into the detonation chamber. This simultaneous expansion results in a complicated wave structure which can be observed in Figure 2b. Here we also note that the dynamically adjustable grid closely follows developing wave structures. In Figure 2c, results are shown at the stage when the two main shock waves generated by the PDE cycle have interacted and are about to leave the computational domain. The maximum pressure here dropped to 1.7 atm.

The computational grid follows the shocks and vortices propagating through the computational domain and we can observe the substantially reduced grid density in the regions of relatively monotonic flow. Figure 2 illustrates the level of detail of this complicated flow regime which can be studied with modern CFD methods and algorithms.

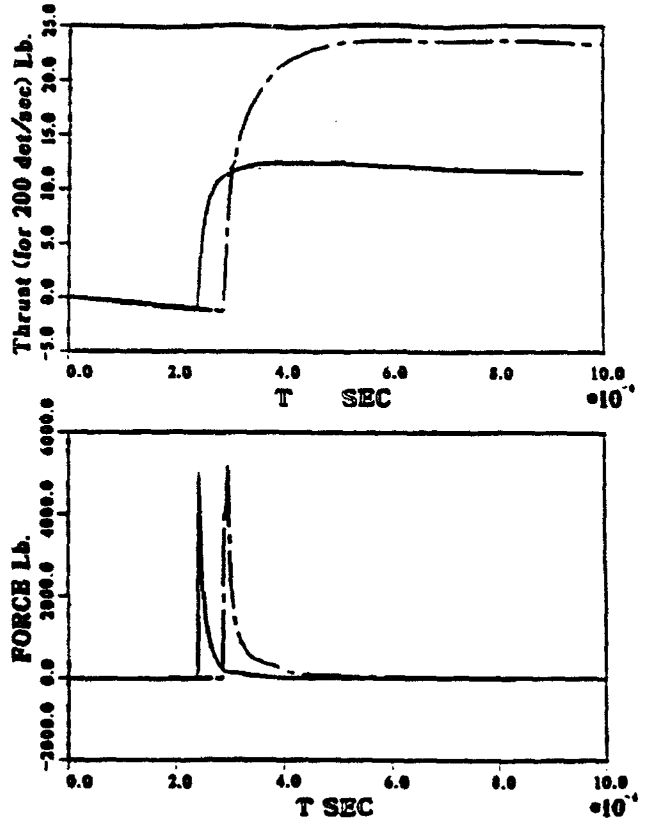


Figure 3. Time averaged thrust and force data from simulation of 8cm (solid lines) and 16cm (dashed lines) detonation chambers, 200 Hz detonation frequency.

In Figure 3 the total force and time averaged thrust generated by the device in the simulations just discussed for 8cm and 16cm long detonation chambers, are shown as a function of time. The time averaged thrust is based on the total time for one cycle defined as 5.0×10^{-3} sec. This time is equivalent to a detonation frequency of 200 Hz. As seen in the figure, initially the force acting on the thrust wall is close to zero. The simulation was run for 2.0×10^{4} sec physical time to establish a flow pattern characteristic of the steady nonreactive flow of ambient air around the detonation chamber. At the time 2.0×10^{4} sec the detonation wave started to propagate from the aft of the chamber. We can see in Figure 3 that the detonation wave reaches the thrust wall at the time 2.45×10^{4} sec (for 8cm case) and 2.9×10^{4} sec (for 16cm case), when a very large force of $\approx 5.0 \times 10^{3}$lb is felt on the end wall of the detonation chamber. This force is a result of the high pressure behind the detonation wave. It rapidly decays to virtually zero level in $\approx 0.5 \times 10^{-4}$ sec in the 8cm case and $\approx 1.0 \times 10^{-4}$ sec in the 16cm case. The maximum force produced on th thrust wall is the same in both cases. The increase o e detonation chamber volume is most noticeable in thrust data. As we can see in Figure 3 the average r ust increases from 12 Lbs in the 8cm chamber ase to 24 Lbs in 16cm chamber case. This res shows that the thrust of the detonation chamber wi ale linearly with an increase in detonation chamber length when the other parameters are kept constant.

5. Supersonic Missile Simulation

In this section we present the results of a full simulation of a generic supersonic missile powered by a PDE. The purpose of this simulation was to study the requirements placed on the PDE air inlets and internal structures that may be needed to produce a well mixed, uniform flow inside the detonation chamber. In addition, the simulations were carried out on the full vehicle in order to account for all wave drag that a real missile produces; the resulting thrust predictions for the simulations are therefore true net thrust values. We show here the results of a successful geometry that satisfies the requirements of choking flow in the inlet throat and uniform predetonation flow in the chamber produced by means of a grill. The missile geometry and computational grid are shown in figures 4a, 4b, and 4c.

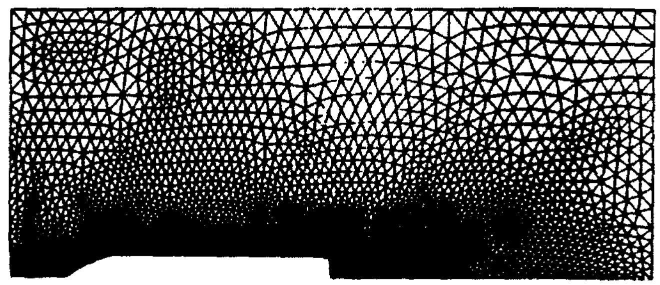


Figure 4a. Unstructured Grid for Missile and Engine Simulation.

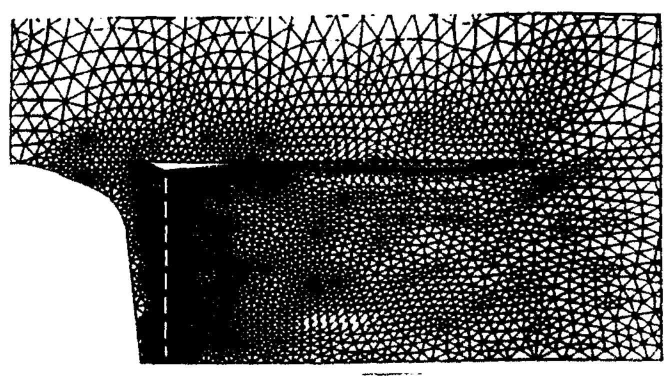


Figure 4b. Grid Detail for Inlet and Manifold.

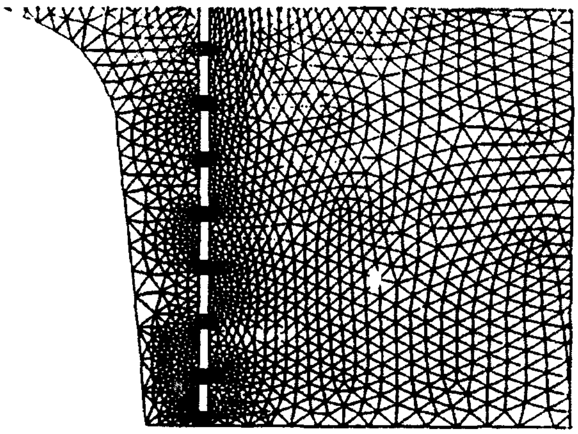


Figure 4c. Grid Detail for Manifold.

Figure 4a shows the main missile body with the PDE covered by the high density of grid points necessary to resolve the details of the PDE chamber, inlets and, grill as shown in the enlarged views of the chamber, figures 4b and 4c.

The simulations were performed by allowing steady subsonic flow conditions to be established in the detonation chamber holding a steady supersonic flow, Mach 2, about the missile. The degree to which this steady and uniform flow can be established in the chamber using the inlet and grill of figure 4 is shown in figure 5. Here the complete flow including the bow shock is shown, figure 5a, as well as an enlarged view of the flow in the vicinity of the inlets showing smaller shocks, figure 5b, and a particle trace showing the streamlines of the uniform chamber flow, figure 5c. When steady flow conditions are reached in the detonation chamber, plane detonation is started at the rear end of the chamber. The detonation then travels towards the inner thrust wall at approximately Mach 4. Figure 6 shows the same sequence of views as figure 5, but with the detonation approximately having travelled halfway to the thrust wall. Notice that the detonation remains more-or-less planar indicating that the flow properties are uniform in the chamber. Figure 7 shows the phenomena associated with the detonation impacting the thrust wall, the high pressure of the detonation wave exhausting from the inlet and particles leaving the chamber through the inlets. The principle results from the simulations of the supersonic missile case are that the use of such a grill structure and inlet shape allow uniform flow to be established before and after detonation in sufficient time that detonation frequencies of 200 cycles per second are obtainable. It is not clear at this time whether such internal grill structures are desirable from the standpoint of structural integritry. This question will be addressed later in planned experimental studies of the PDE.

6. Conclusions

The simulation of the PDE presented in this paper are partial results from an ongoing SAIC research program aimed at development of a practical PDE engine for a wide spectrum of applications including small UAV's and PENAID missiles among others. The primary focus of the results presented here is the scaling of PDE performance with respect to size variation and the establishment of uniform subsonic flow conditions in the detonation chamber before and after detonation.

The results of the scaling studies described in the text lead to scaling laws that can be used to predict the performance of PDE's over some range of parameters assuming that other parameters are held fixed. For example, holding the external Mach number and basic chamber and inlet geometry fixed suggests that the thrust at constant specific fuel consumption produced by the PDE scales as:

$$Thrust = T_1 * \left(\frac{\nu}{\nu_1}\right) * \left(\frac{f}{f_1}\right),$$

where T_1, (v/v_1) and (f/f_1) are the thrust computed for a chamber of volume v_1 operating at frequency f_1, the ratio of a new volume to v_1 and the ratio of the new frequency to f_1 respectively. Thus, thrust should scale linearly with the parameter (v/v_1) * (f/f_1) over some range of this parameter. Departure from this linear variation may occur due to the following reasons: First, since volume is proportional to the product of cross- sectional area and length, $v \sim r^2 l$, ($r \sim$ detonation chamber radius, $l \sim$ chamber length) physical limits will be placed on r and l; if r is too small (less than 1 cm) a detonation will not be sustainable and if l is too small (less than 10 cm) it may be difficult to mix fuel and air effectively. Using the thrust relation established above, we make the following observations. For a PDE device producing 100 pounds thrust at 100 Hz, doubling the frequency and increasing the volume by a factor of 5 yields a thrust level of 1000 pounds. Assuming that the aspect ratio of the chamber (chamber length to radius) is fixed, this would required an engine only 25.5 cm in diameter and 25.5 cm in length. Similarly, scaling the engine down in size to a 5 cm diameter, 5 cm length detonation chamber operatin at 100 Hz yields thrust levels of the order of 3.7 pounds. Of course, the derive relation between thrust and (v/v_1) * (f/f_1) cannot be believed over too wide a range of parameters; but, it does serve to point out the

flexibility in scaleup or scaledown permitted by the PDE concept.

We further conclude that the performance computed for PDEs is encouraging from the point of view of thrust, thrust control, simplicity of the device (no moving parts) and specific fuel consumption (SFC). The specific fuel consumption computed from our simulations ($\sim$ 1Lb/$hr./lb$) is competitive with present day small turbojets (SFCs for small turbojets are in the range of 1.8-2.0 lb./(lb.*hr.)). Thus, for a given mission and vehicle, a PDE propulsion unit could be more fuel efficient resulting in increased range. Moreover, if the expected thrust control in PDEs is realizable, it may be possible to produce propulsion units that can slow down, loiter and maneuver and finally regain full thrust within the time it takes to increase the detonation frequency.

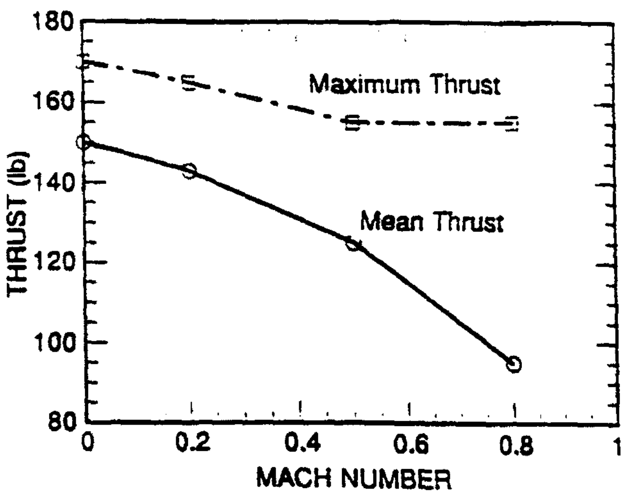


Figure 8. Thrust versus Mach number variation obtained from simulation data.

Another result from the scaling sltudies is that the thrust data show a dependence on the external flight conditions, e.g. Mach number. The Mach number plays a role in the wave drag that the geometry of the PDE will incur; the details of the valve and inlet configurations figure prominently in the total wave drag.

On the other hand, the simulations showed that the timing of the fresh air refilling required to recharge the chamber for subsequent detonations is a strong function of the details of the valve and inlet geometry, the expansion of the combustion products, the resulting over-expansion of the chamber flow and the external flow regime and interaction of the external flow with the internal flow. For subsonic flight, Mach 0.2-0.9, the fresh air entering the chamber comes from two separate principle flow processes: one comes from the flow through any valve or inlet and the other comes from the self- aspiration or reverse flow from the aft end of the chamber due to strong over-expansion. All these processes are interdependent and, in order to search for a given performance in a given device, requires variation of many parameters. The simulation results obtained to date provide an understanding of the effects caused by variation of the above-mentioned parameters and, with the information available, we are able to conclude that a PDE propulsion unit can be optimized (although no optimization studies were carried out) for a given flight regime. For example, if we consider the simulations obtained for constant (number and inlet) geometry but at Mach numbers 0.8, 0.5, 0.2, and 0.0 respectively, the variation of maximum time averaged thrust and mean thrust as a function of Mach number can be characterized as shown in Figure 8.

The decrease in thrust with Mach number has been described earlier[3] to be a result of the increased wave drag produced by the inlet geometry. Optimization of the inlet geometry could help in eliminating a large part of the wave drag. The data contained in Figure 8 could be used to determine the detonation frequency at a given Mach number yielding constant thrust. For a constant thrust level of 90 pounds, the required detonation frequency varies from 84 Hz at $M = 0.0$ to 140 Hz to $M = 0.8$. In a similar fashion, parametric variations of other important aspects of PDE performance, such as minimum time for refill at given Mach number as a function of air inlet opening, can be obtained. In order to find an optimum configuration satisfying given performance over a wide flight regime, a more extensive simulation study will be required. It was mentioned earlier that the simulations presented here were carried out under the assumption of inviscid flow; boundary layer effects were not included. The addition of boundary layers to the PDE engine inlets and valves, the only components where boundary layers will be significant, will lead to increased performance. Roughly the same amount of fresh air will flow into the over-expanded detonation chamber but at a somewhat slower rate and in a pattern that will promote enhanced circulation and hence fuel/air mixing.

A final conclusion can be made concerning the application of PDE's to supersonic vehicles. As shown in the simulations the ability to refill the detonation chamber with fresh air charge is a very strong function of valve and inlet geometry. Refilling may also be somewhat enhanced by the self-aspiration effect, but; to a much less extent than in the subsonic case. The example of supersonic operation discussed in Section 5 shows that care must be taken in design of the inlet or valve configuration. The flow in the chamber must allow for refill and fuel/air mixing. More than likely choked flow conditions will be required at the inlet entrance to the chamber. This could lead to complications in the design of a PDE with simple geometry; choked flow conditions are a function of the external Mach number and a fixed

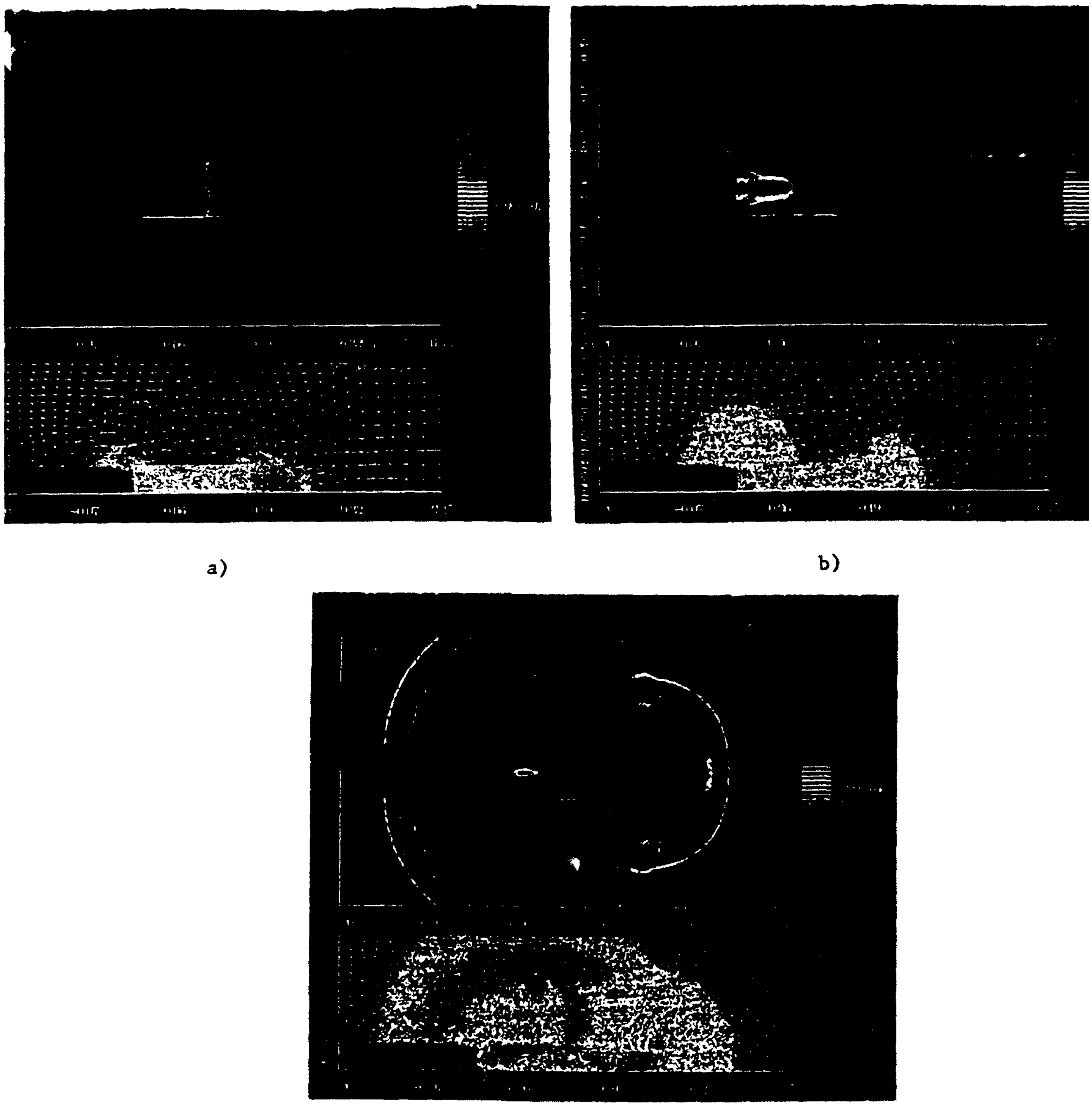
inlet will be optimal only for a small range of the operating envelope. On the other hand, if a given vehicle is to fly at supersonic speeds and is launched at supersonic speeds, this problem may not appear. Further, if the given vehicle is launched at subsonic speeds and a booster is used to bring it up to the required supersonic operating speed, the problem may again not appear. We conclude that the PDE has potential for the supersonic flight regime and it is not excluded that a configuration can be found which will operate over the flight regimes $0.2 <$ Mach number < 3 in a fuel efficient manner.

Acknowledgements

The authors would like to express appreciation to Dr. Adam Drobot of SAIC for helpful suggestions and advice during the course of this work. The work reported on here was partially supported by DARPA under AFOSR contract FY9620-89-(-0087). The authors would like to thank Dr. L. Auslander for his encouragement and interest in this project.

References

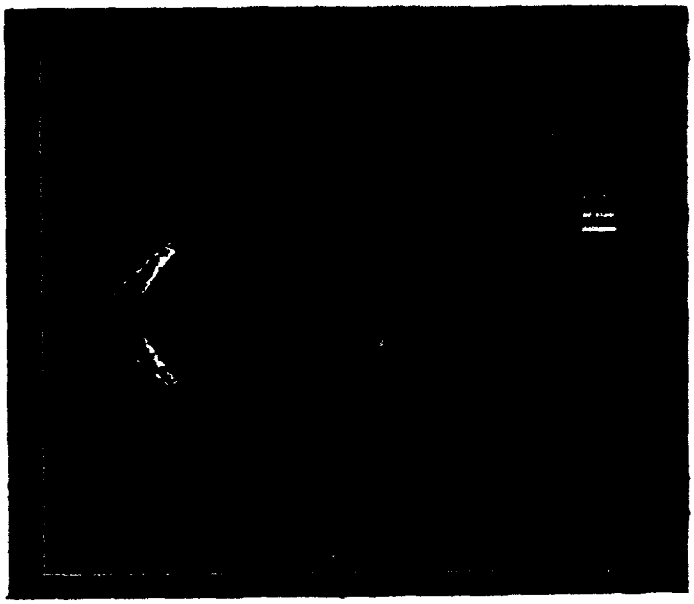
1. Helman, D., Shreeve, R. P., and Eidelman, S., "Detonation Pulse Engine,", AIAA-86-1683, 24^{th} Joint Propulsion Conference, Huntsville, 1986.
2. Eidelman, S., W. Grossmann, I. Lottati, "A Review of Propulsion Applications of the Pulsed Detonation Engine Concept," AIAA 89-2466, AIAA/ASME/SAE/ASEE 25th Joint Propulsion Conference, Monterey, CA, July 10-12, 1989 (to be published in AIAA Journal of Propulsion).
3. Eidelman, S., W. Grossmann, and I. Lottati, "Computational Analysis of the Pulsed Detonation Engines and Applications," AIAA 90-0460, 28th Aerospace Sciences Meeting, Reno, NV, Jan 8-11, 1990.
4. Shultz-Grunow, F., "Gas-Dynamic Investigation of the Pulse-Jet Tube," NACA TM-1131, Feb. 1947.
5. Zinn, B. T., Miller, N. Carvelho, J. A., and Daniel B. R.,"Pulsating Combustion of Coal in Rijke Type Combustor," 19th International Symposium on Combustion, 1197-1203, 1982.
6. Lottati, I., S. Eidelman, A. Drobot, "A Fast Unstructured Grid Second Order Godunov Solver (FUGGS)," Paper AIAA 90-0699, 28th Aerospace Sciences Meeting, Reno, NV, Jan 8-11, 1990.
7. Lottati, I., S. Eidelman, A. Drobot, "Solution of Euler's Equations on Adaptive Grids Using a FUGGS," to be published in Proceedings of Second International Conference on Free-Lagrange Methods, Held at Jackson Hole, June 1990.
8. Eidelman, S. Collela, P., and Shreeve R. P., "Application of the Godunov Method and Its Second Order Extension to Cascade Flow Modeling,", AIAA Journal v. 22, 10 (1984).



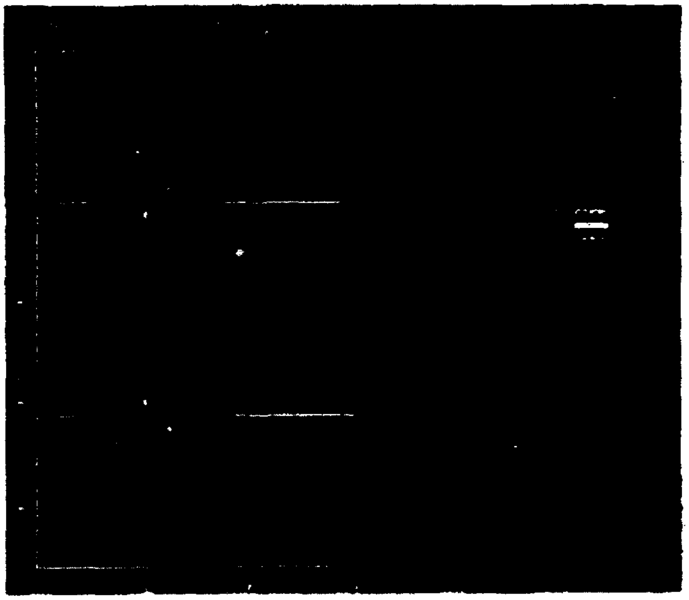
a) b)

c)

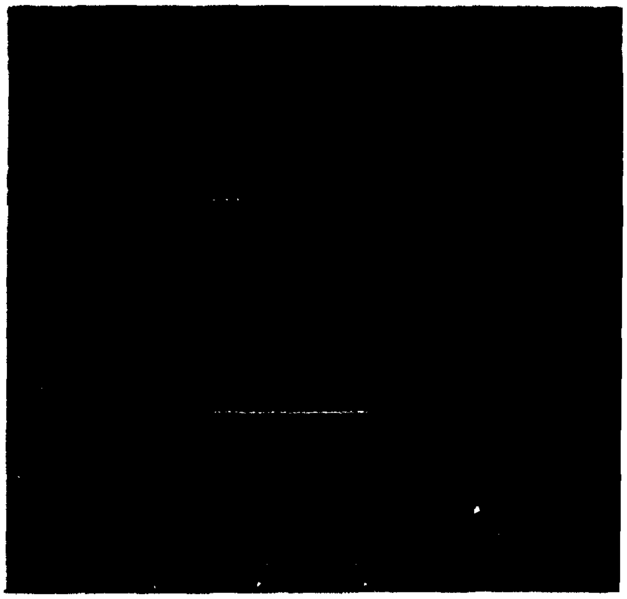
Figure 2. Pressure contours and computational grid for 16 cm long PDE. External flow M = 0.8.



a. Pressure Contours. Missile and Engine.

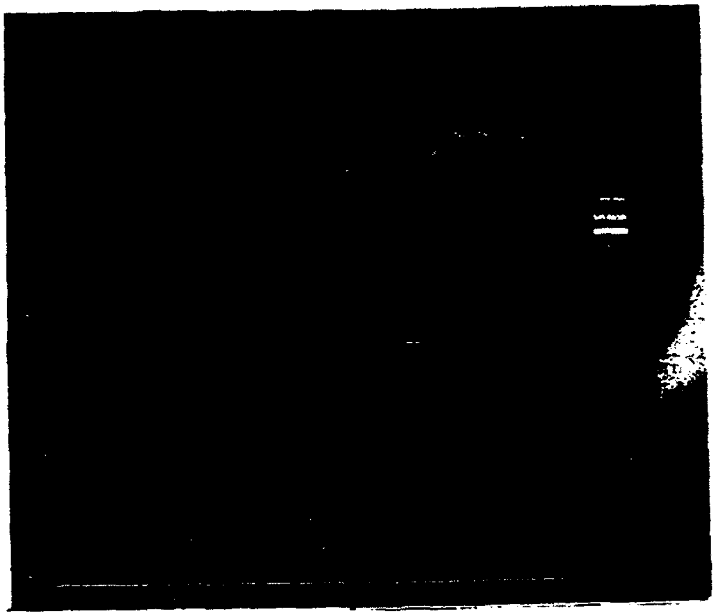


b. Pressure Contours. Detonation Engine.

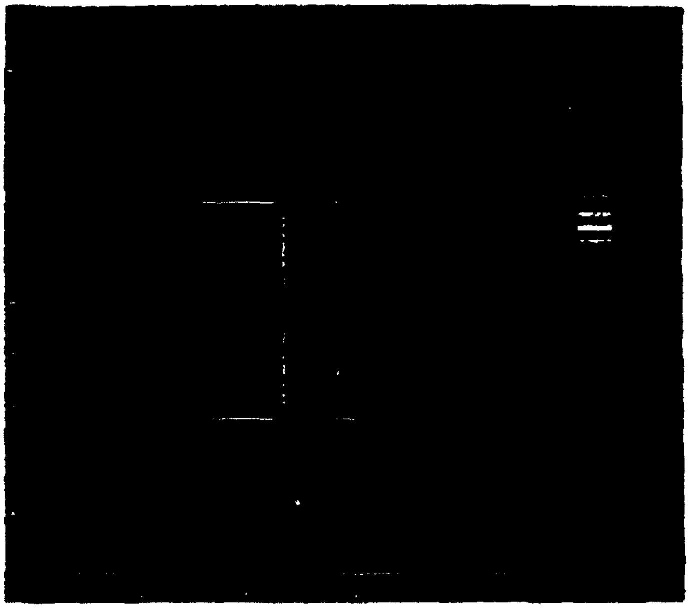


c. Traced Particles. Detonation Engine.

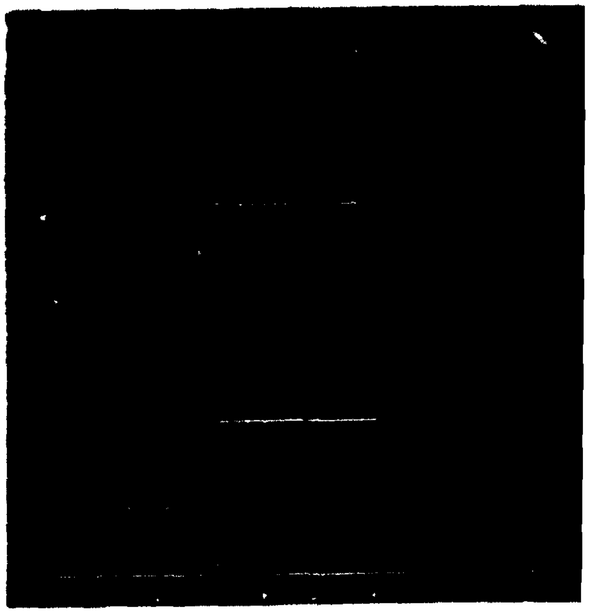
Figure 5. Supersonic missile simulation. Missile speed M = 2.0. Time t = 0.0.



a. Pressure Contours. Missile and Engine.

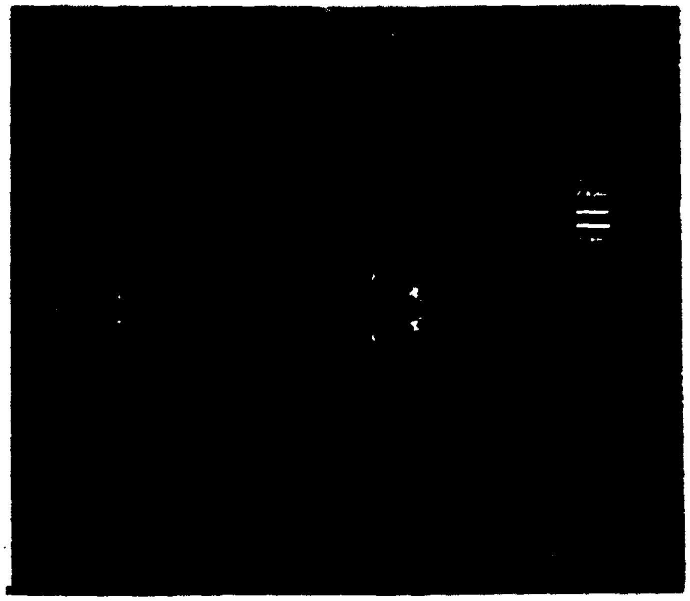


b. Pressure Contours. Detonation Engine.



c. Traced Particles. Detonation Engin

Figure 6. Supersonic missile simulation. Missile speed M = 2.0. Time $t = 2.0 \cdot 10^{-5}$ sec.



a. Pressure Contours. Missile and Engine.

b. Pressure Contours. Detonation Engine.

c. Traced Particles. Detonation Engine.

Figure 7. Supersonic missile simulation. Missile speed M = 2.0. Time t = $2.0 \cdot 10^{-4}$ sec.

Surface and Coatings Technology, 49 (1991) 387-393

Plasma enhanced chemical vapor deposition modeling

E. Hyman, K. Tsang, I. Lottati and A. Drobot
Science Applications International Corporation, 1710 Goodridge Drive, McLean, VA 22102 (U.S.A.)
B. Lane*, R. Post and H. Sawin†
Applied Science and Technology, 35 Cabot Road, Woburn, MA 01801 (U.S.A.)

Abstract

We are developing a model to simulate the plasma enhanced chemical vapor deposition (PECVD) of thin diamond films. The emphasis to date has been on the development of stand-alone modules to simulate the microwave-induced time-dependent electric and magnetic fields, the generation and energization of plasma electrons in the discharge, the non-equilibrium hydrocarbon chemistry, and the development of a two-dimensional unstructured mesh hydrodynamics solver capable of simulating flow through geometrically realistic reactors. The coupling of the various modules, and the incorporation of a surface chemistry module for the substrate deposition, into a self consistent reactor model is underway. We present some preliminary results from components of a model 2.45 GHz microwave reactor employing H_2 with 1% CH_4 and operating at a gas pressure of 5.3×10^3 Pa (40 Torr). We have completed an electromagnetic model of the microwave energy deposition in the plasma and calculated the field patterns in the reactor. We have also performed point calculations of the time-dependent electron distribution and of the build-up of atomic hydrogen, the gas temperature, and the resulting generation of CH_3, C_2H_2, and other hydrocarbon radicals. We have also completed a fluid simulation of the flow through the reactor using unstructured mesh techniques. The results we discuss in this paper indicate that careful treatment of non-equilibrium processes in PECVD reactors as well as accurate representation of reactor geometry are essential to a useful simulation capability.

1. Introduction

The ability to deposit thin diamond films rapidly onto substrates with a high degree of uniformity using the plasma enhanced chemical vapor deposition (PECVD) technique is a high priority technology goal. It is generally recognized that an improved understanding of the microscopic mechanisms in PECVD reactors and of the sensitivity of the various reactor parameters is needed. The important design issues for PECVD reactors are as follows: efficient coupling of microwave energy to the plasma and to the process gas; efficient transport of activated process gas to the wafer or substrate deposition area; efficient use of the injected gas; uniformity of chemically active species flux across the deposition area. It is desirable to avoid reactor designs that have the following: high microwave electric fields in regions away from the desired plasma formation location, leading to plasma discharge near chamber walls or breakdown of dielectric materials; flow patterns which carry activated species to the reactor walls or out through pumping ports rather than to the wafer; stagnant or circulating flow patterns above the wafer, buffering the wafer from the desired chemically active species.

To understand these issues and to provide input to improved reactor designs we are developing a self-consistent numerical model which simulates each of the essential mechanisms in the PECVD reactor. A physically realistic model requires careful simulation of the electromagnetics, the plasma physics, the neutral gas flow, and the homogeneous and heterogeneous chemistry. Furthermore, the different elementary processes in the reactor are highly interactive; for this reason it is difficult to foresee intuitively the impact of varying one or another reactor parameter. For example, the microwave source induces a complex, geometrically dependent and time varying electric field which ionizes the gas; the resultant build-up of electrons alters the developing electric field distribution. The microwave energy input heats the electrons, and the energetic part of the non-Maxwellian electron energy distribution dissociates the gas, inducing a rise in the gas temperature. The amount of dissociation and heating depends sensitively on the high energy tail of the electron distribution which consequently must be accurately determined. The interaction of the neutral gas with the plasma alters the molecular input stream of H_2 to include a substantial

*Permanent address: MIT, Plasma Fusion Center, Cambridge, MA 02139, U.S.A.

†Permanent address: MIT, Chemical and Electrical Engineering Department, Cambridge, MA 02139, U.S.A.

0257-8972/91/$3.50

component of atomic hydrogen, and this, in turn, affects the ionization rate and the electron distribution. The hydrocarbon chemistry is non-equilibrium and both the flux and the spatial distribution of appropriate radicals reaching the wafer are very sensitive to the geometrical configuration of the reactor and to the details of the flow configuration through the reactor.

We have focused to date on the development of modules to simulate the microwave-induced time-dependent electric and magnetic fields, the generation and energization of plasma electrons in the discharge, the evolution of the molecular and atomic hydrogen gas, the non-equilibrium hydrocarbon chemistry, and the development of a two-dimensional unstructured mesh hydrodynamics solver capable of simulating flow through geometrically realistic reactors. The coupling of these modules, and the incorporation of a surface chemistry module for the substrate deposition, into a self consistent reactor model is underway. In the next section we describe in some detail the generation of the microwave field and the transfer of the field energy to the electrons. This is followed by preliminary model results and our conclusions.

2. Microwave field and plasma generation

The absorption of microwaves and the creation of the plasma which transfers energy to the neutral species in the reactor involves the solution of two closely coupled problems. They are (1) the determination of the electromagnetic field patterns in the complex geometry of the reactor and (2) the formation of the electron distribution function. At a pressure of 5.3×10^3 Pa (40 Torr) and a gas temperature in the plasma region greater than 2000 K, the mean free path of an electron with neutrals is approximately 5×10^{-5} m. During the time an electron gains the average electron energy (approximately 2 eV) typical of the reactors we are modeling, it undergoes around 150 collisions and has a mean displacement of approximately 7×10^{-4} m. Thus, to an excellent approximation, the heating of the electrons results from the microwave electric fields which are local to the electron's spatial location.

The electron distribution function satisfies the Boltzmann equation. Because an electron undergoes many collisions as it is heated, the distribution function is nearly isotropic and can be well approximated by the zero and first order terms of a spherical expansion, the latter representing a distortion of the distribution function in the direction of the applied field, oscillating at the microwave frequency ω. The equation for the electron distribution function is

$$\frac{1}{3}\left[\left(\frac{eE_0}{m_e}\right)^2 \frac{1}{v^2}\frac{\partial}{\partial v}\left(v^2 \frac{\nu_m}{\nu_m^2+\omega^2}\frac{\partial F_0}{\partial v}\right) + v^2\nabla\cdot\left(\frac{1}{\nu_m}\nabla F_0\right) + v\nabla\cdot\left(V\frac{\partial F_0}{\partial v}\right)\right]$$
$$= L_i + L_x - \frac{2m_e}{M}\frac{1}{v^2}\frac{\partial}{\partial v}(v^3\nu_m F_0)$$

where E_0 is the amplitude of the electric field, e, m_e, and v are the electron charge, mass, and velocity respectively, ν_m is the electron momentum transfer frequency, F_0 is the zero order approximation to the distribution function, V is the bulk fluid velocity, M is the neutral mass, and L_i and L_x are the inelastic loss terms that affect the distribution function respectively via ionization and excitation of rotational, vibrational, and electronic levels. The first term on the left represents the electron velocity diffusion due to the cumulative affect of many small angle scatterings of the electron induced by the oscillating electric field. The next two terms give the affect of the divergence of the diffusive and convective fluxes respectively. The last term on the right gives the energy loss due to elastic collisions.

Below a critical electric field the ionization rate is exceedingly small and hence the electron density and power deposited per unit volume are also small. Above the critical field the ionization rate and power deposition increase rapidly. If the power deposition is kept approximately constant and equal to the power injected into the reactor the electric field will rapidly adjust to a level close to the breakdown value. The power deposited per unit volume scales as $E_0^2 n_e$, where n_e is the electron density; as the electron density rises the electric field drops. The above considerations provide the necessary prescription for determining the time-dependent evolution of the electric field, the electron density, and the electron energy distribution. Using a set of elastic and inelastic cross-sections, the last two parameters define the time-dependent evolution of the fluid, including the build-up of atomic hydrogen and the rise in the gas temperature as the gas dissociates. In addition to H_2 and H, the Boltzmann calculation monitors the evolution of H_2^+, H_3^+, H^+, and H^- and separately tracks each of the three lowest vibrational levels of H_2. The hydrocarbon chemistry is initiated by energetic electrons, but being trace constituents the hydrocarbons do not significantly affect the electron development. It is useful to take advantage of the separation of time scales inherent in this problem. The electron distribution function is established on a time scale of around 10^{-8} s, the electron density growth occurs over approximately 1 μs, and the hydrogen dissociation and hydrocarbon chemistry as well as fluid convection and diffusion occur on a millisecond time scale.

3. Results

We present calculations from the modules of the PECVD model that have been constructed and tested. The results obtained are designed to identify important physical mechanisms and to determine the regimes where they are critical. The calculation of the electric and magnetic fields was accomplished using SAIC's MASK code, a general two-dimensional electromagnetic code designed for the study of microwave devices of arbitrary geometrical configuration. The code introduces the electric fields at the input port and allows them to propagate into the reactor, which can include arbitrarily shaped regions of dielectric or conducting bodies. It employs a finite difference representation of the full set of time-dependent Maxwell equations and solves the initial value problem. Figure 1 shows the results of a simulation for a generic reactor in which the plasma is modeled as a spherical shell with a finite conductivity and a finite dielectric constant. Ultimately the plasma model will be replaced by results of Boltzmann calculations over the plasma region. The MASK calculation retains both the field strength and the phase dependence and determines the energy deposition in the target plasma. The results shown are contours of constant field amplitude for the axial (a) and radial (b) electric components in an azimuthally symmetric configuration. The bottom horizontal line is the axis of symmetry. The input wave is introduced on the left and propagates into the reactor region through a radially expanding transition region. The substrate is on the right in the figure and immediately above it is the outlet for the reacting gases. The figure indicates regions of high and low field concentration which will provide an important tool for reactor design. This calculation determines the energy deposition in the plasma and, hence, the reactor efficiency. When performed self consistently it also predicts the shape of the plasma region and the subsequent coupling to the hydrodynamic calculation.

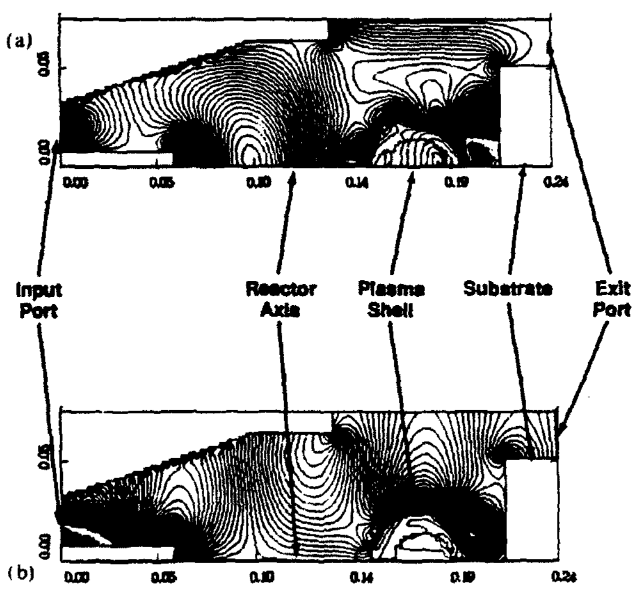


Fig. 1. MASK calculation of (a) the axial and (b) the radial components of the electric field in a model reactor. Shown in the figure are contours of constant amplitude. The reactor axis is the bottom horizontal line. The substrate shelf is at the right and the outlet for the reacting gases is above it. The plasma shell is centered on the axis to the left of the substrate.

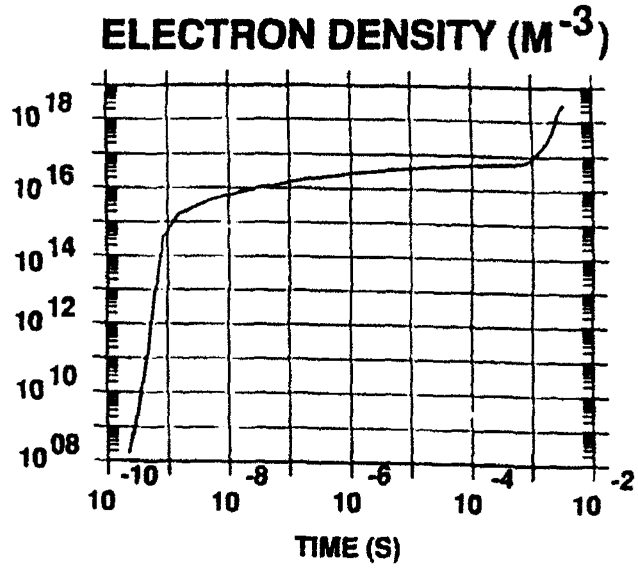
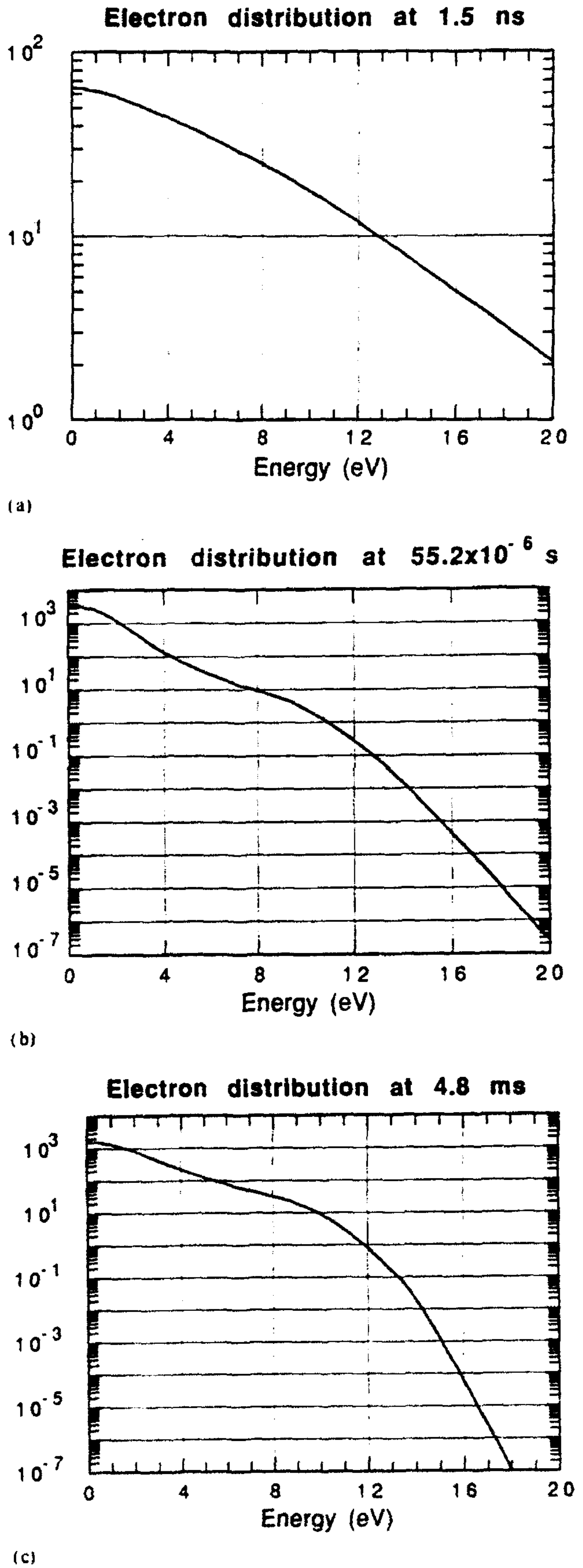


Fig. 2. Time development of the electron density at a point in the reactor of high electric field corresponding to the one-point simulation described in the text.

The Boltzmann module calculation simulates the electron and heavy particle evolution at a location within the reactor where the electric field is sufficiently high to create and sustain a plasma. We maintain a constant deposited microwave power and assume the gas pressure is kept constant at 5.3×10^3 Pa (40 Torr). The simulation runs for several milliseconds, beyond which time advection and diffusion effects, not included in this calculation, would become important. The initial conditions are $[H_2] = 1.28 \times 10^{24}$ m^{-3}, $[CH_4] = 1.28 \times 10^{22}$ m^{-3}, gas temperature $T = 300$ K. Figure 2 shows the electron density rising rapidly to approximately 10^{15} m^{-3} in around 1 ns, a result of the very energetic electron distribution at early times. Thereafter, it increases nearly two more decades over about 10^{-5} s, the slower increase being reflective of the less energetic electron spectrum as the electrons give up energy to the various inelastic processes. The increase on a millisecond time scale is associated with the conversion of the gas from the molecular to the atomic state which leads to an increasing fraction of atomic ions (which recombine much more slowly than molecular ions) and also causes



an adjustment in the electron distribution function. The evolution of the electron distribution function as determined by the Boltzmann equation is illustrated for these three time regimes in Fig. 3 in which the ordinate scale is arbitrary. In Fig. 3(a) at 1.5 ns we see the very energetic electron spectrum: the average electron energy is around 8 eV. In Fig. 3(b) at 55.2 μs the average energy has dropped to 2 eV and in Fig. 3(c) at 4.8 ms when atomic hydrogen predominates, the spectrum has changed again, the average electron energy increasing moderately to about 2.7 eV.

The evolution of the hydrocarbon species is simulated with a chemistry code that uses the output of the Boltzmann code. The reactions used in the hydrocarbon model are listed in Table 1 along with the constants A, b, and E which determine the rate coefficient k according to $k = AT^b \exp(-E/T)$. The code calculates the rate for each reverse reaction that is not known, using detailed balance. The hydrocarbon chemistry is initiated by the electrons which dissociate H_2 (and also the CH_4), causing the release of chemical energy and heating the gas. Figure 4 shows the gas temperature as a function of time for the simulation described above. In Fig. 5 we show the evolution of 12 hydrocarbon species plus H_2 and H out to 3 ms, at which time the H_2 and H densities are approximately equal. Although the formation of H is initiated by the electron dissociation of H_2, after about 2.5 ms with rise in temperature, thermal dissociation of H_2 becomes predominant. The build-up of CH_3 due to the dissociation of CH_4 occurs very early (approximately 30 μs) but it reacts with itself to form C_2H_6 and drops to a local minimum before 1 ms. As the temperature increases, however, the CH_3 recombination reaction rate decreases and CH_3 increases to a new maximum near 2 ms; thereafter it decreases once more as the CH_4 becomes exhausted. Acetylene (C_2H_2) results from the chain of reactions initiated by the formation of C_2H_6, thence to C_2H_5 and, in turn, to C_2H_4, C_2H_3 and finally to C_2H_2 which persists to the end of the simulation. In general, the hydrocarbon species do not have time to reach the equilibrium values that the gas temperature would dictate. Thus, the time between their formation in the plasma and their reaching the substrate determines the densities of the critical radical species reaching the substrate.

Fig. 3. Electron energy distribution for the simulation as in Fig. 2. (a) at a time before inelastic processes reduce the average electron energy; (b) at an intermediate time when the average electron energy is about 2 eV; (c) at a time when dissociation of the H_2 is nearly complete. The ordinate scale is arbitrary.

TABLE 1. Hydrocarbon reactions and rate coefficients

Reaction	$A(10^{6(n-1)}(m)^{3(n-1)}s^{-1})^a$	b	E(K)	Range(K)
$H + H + H_2 \rightarrow H_2 + H_2$	2.7×10^{-31}	−0.6	0	100–5000
$H_2 + H_2 \rightarrow H + H + H_2$	1.5×10^{-9}	0	4.84×10^4	2500–8000
$CH_4 + H \rightarrow CH_3 + H_2$	3.6×10^{-20}	3.0	4.40×10^3	300–2500
$CH_3 + H_2 \rightarrow CH_4 + H$	1.1×10^{-21}	3.0	3.90×10^3	300–2500
$CH_4 + \genfrac{}{}{0pt}{}{H_2}{CH_4} \rightarrow CH_3 + H + \genfrac{}{}{0pt}{}{H_2}{CH_4}$	$\genfrac{}{}{0pt}{}{(2.9)}{(18.6)}3.3 \times 10^{-7}$	0	4.45×10^4	1500–3000
$CH_3 + H + \genfrac{}{}{0pt}{}{H_2}{CH_4} \rightarrow CH_4 + \genfrac{}{}{0pt}{}{H_2}{CH_4}$	$\genfrac{}{}{0pt}{}{(2.9)}{(18.6)}2.2 \times 10^{-21}$	−3.0	0	300–2500
$CH_3 + CH_3 \rightarrow C_2H_5 + H$	1.3×10^{-9}	0	1.34×10^4	1500–3000
$C_2H_5 + H \rightarrow CH_3 + CH_3$	5.0×10^{-11}	0	0	300–1500
$CH_3 + CH_3 \rightarrow C_2H_4 + H_2$	1.7×10^{-8}	0	1.61×10^4	1500–2500
$CH_3 + \genfrac{}{}{0pt}{}{H_2}{CH_4} \rightarrow CH_2 + H + \genfrac{}{}{0pt}{}{H_2}{CH_4}$	$\genfrac{}{}{0pt}{}{(2.9)}{(18.6)}1.7 \times 10^{-8}$	0	4.56×10^4	1500–3000
$CH_2 + H \rightarrow CH + H_2$	6.6×10^{-11}	0	0	300–2500
$CH_2 + CH_3 \rightarrow C_2H_4 + H$	6.6×10^{-11}	0	0	300–2500
$C_2H_6 + H \rightarrow C_2H_5 + H_2$	9.0×10^{-22}	3.5	2.62×10^3	300–2000
$C_2H_6 + CH_3 \rightarrow C_2H_5 + CH_4$	9.1×10^{-25}	4.0	4.17×10^3	300–2000
$C_2H_6 + \genfrac{}{}{0pt}{}{H_2}{CH_4} \rightarrow CH_3 + CH_3 + \genfrac{}{}{0pt}{}{H_2}{CH_4}$	$\genfrac{}{}{0pt}{}{(2.9)}{(18.6)}1.7 \times 10^{-5}$	0	3.43×10^4	800–2500
$C_2H_5 + C_2H_5 \rightarrow C_2H_4 + C_2H_6$	2.3×10^{-12}	0	0	300–1200
$C_2H_5 + \genfrac{}{}{0pt}{}{H_2}{CH_4} \rightarrow C_2H_4 + H + \genfrac{}{}{0pt}{}{H_2}{CH_4}$	1.7×10^{-7}	0	1.56×10^4	700–1500
$C_2H_4 + H \rightarrow C_2H_3 + H_2$	2.5×10^{-10}	0	5.14×10^3	700–2000
$C_2H_4 + \genfrac{}{}{0pt}{}{H_2}{CH_4} \rightarrow C_2H_2 + H_2 + \genfrac{}{}{0pt}{}{H_2}{CH_4}$	$\genfrac{}{}{0pt}{}{(2.9)}{(18.6)}4.3 \times 10^{-7}$	0	3.99×10^4	1500–2500
$C_2H_4 + \genfrac{}{}{0pt}{}{H_2}{CH_4} \rightarrow C_2H_3 + H + \genfrac{}{}{0pt}{}{H_2}{CH_4}$	$\genfrac{}{}{0pt}{}{(2.9)}{(18.6)}4.3 \times 10^{-7}$	0	4.86×10^4	1500–2500
$C_2H_4 + CH_3 \rightarrow C_2H_3 + CH_4$	7.0×10^{-13}	0	5.59×10^3	300–1000
$C_2H_3 + H \rightarrow C_2H_2 + H_2$	3.3×10^{-11}	0	0	300–2500
$C_2H_3 + \genfrac{}{}{0pt}{}{H_2}{CH_4} \rightarrow C_2H_2 + H + \genfrac{}{}{0pt}{}{H_2}{CH_4}$	$\genfrac{}{}{0pt}{}{(2.9)}{(18.6)}5.0 \times 10^{-9}$	0	1.61×10^4	500–2500
$C_2H_2 + H + \genfrac{}{}{0pt}{}{H_2}{CH_4} \rightarrow C_2H_3 + \genfrac{}{}{0pt}{}{H_2}{CH_4}$	$\genfrac{}{}{0pt}{}{(2.9)}{(18.6)}1.11 \times 10^{-30}$	0	3.5×10^2	300–500
$C_2H_2 + H \rightarrow C_2H + H_2$	1.0×10^{-10}	0	1.19×10^4	300–3000
$C_2H + H_2 \rightarrow C_2H_2 + H$	2.5×10^{-11}	0	1.56×10^3	300–3000
$C_2H_2 + CH_2 \rightarrow C_3H_3 + H$	3.0×10^{-12}	0	0	>298
$C_2H_2 + C_2H \rightarrow C_4H_2 + H$	5.8×10^{-11}	0	0	300–2500
$C_2H_2 + \genfrac{}{}{0pt}{}{H_2}{CH_4} \rightarrow C_2H + H + \genfrac{}{}{0pt}{}{H_2}{CH_4}$	$\genfrac{}{}{0pt}{}{(2.9)}{(18.6)}6.6 \times 10^{-8}$	0	5.38×10^4	1500–3500

[a] n denotes the number of reactants.

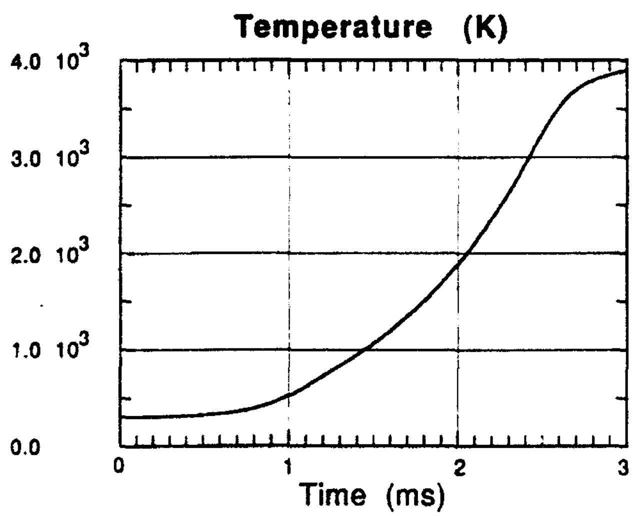
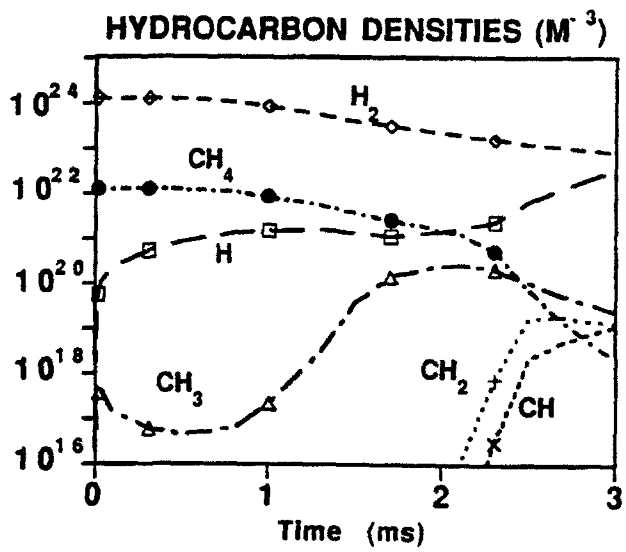
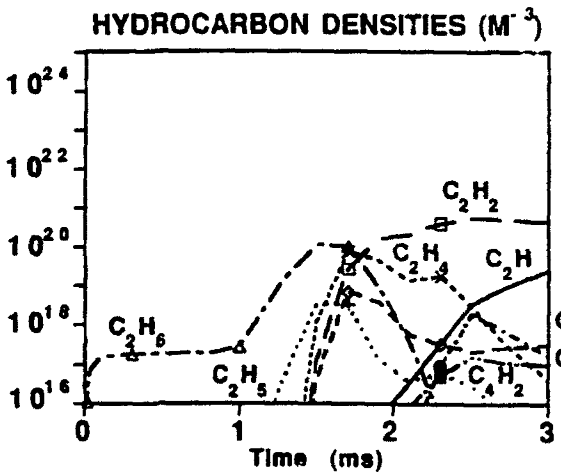


Fig. 4. Evolution of the gas temperature for the simulation as in Fig. 2.

Finally, we present preliminary results of a fluid simulation of a generic PECVD reactor, using SAIC's FUGG code which is capable of performing fluid calculations over arbitrarily complex geometries. The code employs an unstructured grid allowing extremely fine resolution in critical areas while employing coarser gridding in regions where quantities vary slowly. The code was designed for the study of flow problems dominated by convection and is presently being modified to incorporate thermal conduction and viscosity effects. In Fig. 6 we show two examples of the code's triangular gridding capability. In both cases a cross-section of the azimuthally symmetric model reactor is shown, where the left vertical boundary represents the reactor axis. Three gas inlet ports are modeled allowing the gas to enter at the top (in Fig. 6(a) the plenum region above the inlet ports is also modeled). The gas exits through the horizontal boundary at the bottom right. The substrate wafer is represented in Fig. 6(a) by the left half of the lower horizontal boundary and in Fig. 6(b) by the shelf at the lower left. In Fig. 6(a) the variable gridding capability is clearly illustrated and, in particular, the fine gridding needed in the inlet ports is shown. Figure 7 shows results of a fluid calculation for the reactor of Fig. 6(b) in which hydrogen gas enters at $50\ m\ s^{-1}$ at a pressure of 5.3×10^3 Pa (40 Torr). A heating source of 1.5 kW over a spherical volume of radius 0.035 m, centered on the reactor axis and mid-



(a)



(b)

Fig. 5. Evolution of the hydrocarbon densities for the simulation as in Fig. 2.

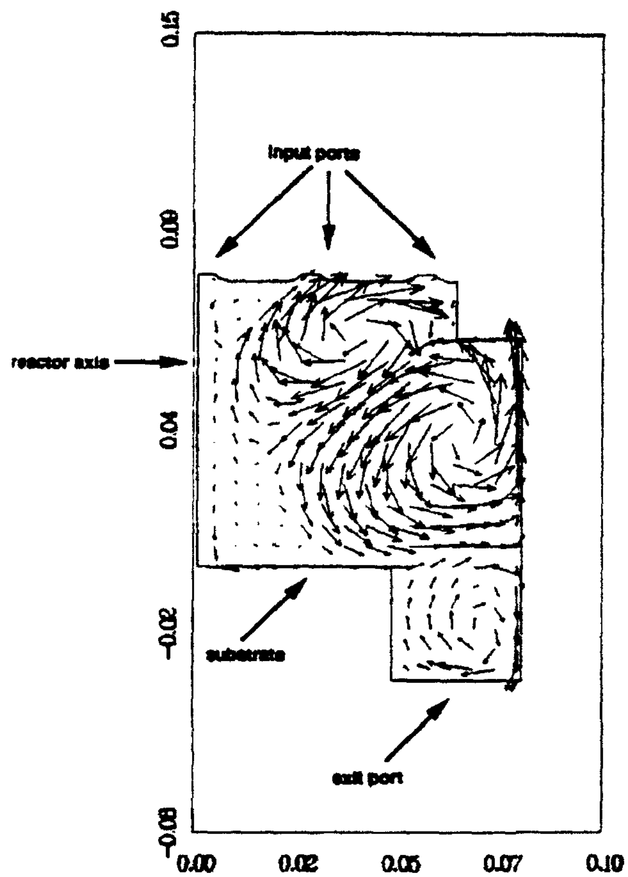


Fig. 7. Velocity field for reactor of Fig. 6(b) in which H_2 enters at a velocity of 50 m s^{-1} at a pressure of 5.3×10^3 Pa (40 Torr). A heating source of 1.5 kW to simulate the effect of the plasma is centered on the axis between the inlet port and the wafer in a spherical volume of radius 0.035 m. The ordinate and abscissa dimensions are in meters.

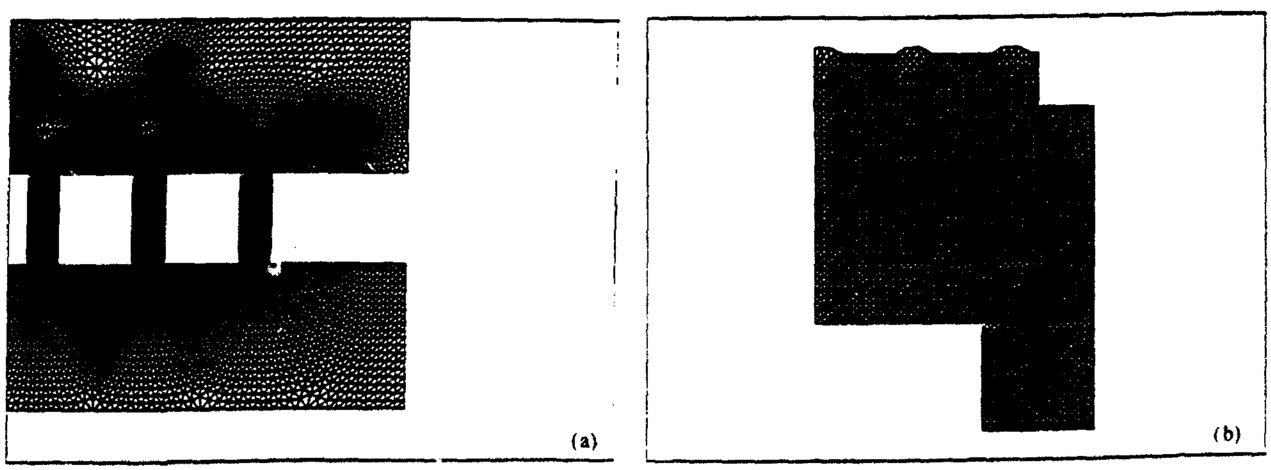


Fig. 6. Examples of the FUGG code's unstructured gridding capability for two model reactors ((a) and (b)).

way between the inlet port and the wafer, is included to simulate approximately the effect of the plasma source. Shown are velocity vectors for the flow 2.6 ms after the plasma is turned on. Also calculated but not shown are the pressure, density, and temperature fields. While conclusions should be tempered because of the current lack of inclusion of thermal effects in the code and because the results represent a transient pre-steady-state stage, the effects of buoyancy are apparent. The complex vortex flows seen suggest this reactor configuration would be very poor for efficient diamond deposition.

4. Discussion and conclusions

We have presented results of a model under development that will permit the simulation of PECVD reactors of arbitrary geometry. The model will be an important tool providing better understanding of the microscopic processes occurring within the reactor, permit parameter studies to identify those parameters which critically affect both the rate of deposition and the uniformity of the deposition over the wafer surface, and ultimately enable the design of improved reactors. We have identified several critical elements in the modeling effort that need to be treated carefully if simulation results are to be meaningful. First, the electromagnetic fields which initiate the plasma formation need to be determined in the realistic reactor geometry, including effects of all metallic, dielectric, and insulator elements actually present, to ensure that the fields are high in the desired plasma formation region but not elsewhere. Second, the coupling of the fields to the plasma electrons, to determine accurately both the time development of the electron density and their energy distribution, is most important for determining the evolution of the rate of hydrogen dissociation and the rise in the gas temperature. This, in turn, critically determines the non-equilibrium hydrocarbon chemistry development, a third area that needs to be carefully modeled. Finally, the flow of hydrocarbon radicals to the wafer is very sensitive to the reactor's geometrical configuration, its thermal properties, and the location of the plasma relative to the wafer. In conclusion, our results emphasize the highly non-equilibrium and coupled nature of PECVD reactor processes and the strong influence of reactor geometry. A numerical simulation that is useful must address all such issues.

Acknowledgment

This research was supported by the U.S. Army Missile Command and sponsored by the Defense Advanced Research Projects Agency (DARPA) under contract DAAH01-90-C-0279 and by the Independent Research and Development Program of Science Applications International Corporation (SAIC).

Nonlinear signal processing using
integration of fluid dynamics equations

by

S. Eidelman, W. Grossmann and A. Friedman*

Science Applications International Corporation
1710 Goodridge Drive
McLean, Virginia 22102

1. INTRODUCTION

Very recently, there have been exploratory efforts in image processing based on nonlinear methods.[1] These efforts involve systems of nonlinear hyperbolic partial differential equations in combination with local wave representation, such as wavelets, for signal enhancement.[2,3,4] Techniques based on Kalman filtering for feature extraction from complex time-evolving scenes, as well as neural network approaches to image analysis and feature identification, can also be shown to involve nonlinear PDE analogies. The use of nonlinear methods, however, is largely unexplored and may provide another level of improvement for image processing.

If the purpose of an image enhancement process is to highlight the edges of an image, then the technique used in the frequency domain is usually highpass filtering. An image can be blurred, however, by attenuating the high-frequency component of its Fourier transform. Since edges and other abrupt changes in the gray levels are associated with high-frequency components, image sharpening can be achieved in the frequency domain by a highpass filtering process, which attenuates the low-frequency without disturbing high-frequency information in the Fourier transform. The primary problem with this technique is that an ideal discontinuity will have an infinite spectrum of frequencies associated with it. When filtering is applied, some frequencies are cut off, leading to a loss of some edges in an image.

It is interesting to observe that in the field of Computational Fluid Dynamics (CFD) similar problems exist in simulating flows with discontinuities. The problem of simulating flows with discontinuities is less forgiving, since an incorrect calculation usually leads to a complete distortion of the flow field. This has led CFD scientists to develop sophisticated algorithms that identify and preserve discontinuities while integrating the flow field in the computational domain. In the image domain, sharpening is usually done by differentiation. The most commonly used methods involve the use of either gradients or second derivatives of the pixel information. Central differencing is usually used to calculate the derivatives. CFD research has shown that this strategy will lead in many cases to a smearing of the flow discontinuities (analog of the image edges in image enhancement).

Here, we describe a new and unique image sharpening method based on computational techniques developed for CFD. Our preliminary experience with this method shows its capability for nonlinear enhancement of image edges as well as deconvolution of an image with random noise. This indicates a potential application for image deconvolution from sparse and noisy data resulting from measurements of backscattered laser-speckle intensity.

*Present address: Brookhaven National Lab., Apton, NY 11973

2. THE CFD IMAGE ENHANCEMENT TECHNIQUE

Considerable attention has been devoted to the development of numerical methods and algorithms for Computational Fluid Dynamics during the last thirty years. In recent years, however, our understanding of numerical algorithms for a particular class of problems in gas dynamics described by the Euler equations has become more complete. The main numerical difficulty in solving invisid compressible flows described by Euler equations is the occurrence of features that, in the invisid approximation, are discontinuous and even in the presence of viscosity are too small to be resolved on an affordable computational mesh. These flow discontinuities in which the fluid state jumps across shock waves or contact surfaces are extremely important in fluid simulations. Most of the efforts in developing numerical techniques in fluid dynamics over the last twenty years were devoted to accurate simulations of these discontinuities. Initially, naive numerical methods that used a formal finite difference representation of the conservation equations on a computational grid were employed. That led to disastrous results, smearing of the discontinuities, and spurious oscillations. Subsequently, sophisticated nonlinear techniques, which allowed accurate simulations of complex discontinuities without smearing and ringing, were developed. These new methods also satisfy a very demanding criteria for robustness and allow simulation of the wide range of flow problems without adjustment or tuning of the numerical technique.

The numerical methods that allow high accuracy resolution of flow discontinuities are so-called TVD (Total Variation Diminishing) methods. The Second Order Godunov Method is one of the most successful numerical techniques developed for this purpose. In Figure 1, an example is given of a solution using the Second Order Godunov Method for a complicated case of multiple shock waves,[5] illustrating the ability of this method to capture and simulate sharp discontinuities.

The Second Order Godunov Method was developed based on an understanding of the phenomenology of signal propagation in the gasdynamical system. The numerical algorithm implementing this method is not analytic and is based on a set of steps that can be considered as wave filters. These filters are designed to not smear the discontinuity (edge), suppress the spurious oscillations, and propagate the relevant signals through the system. The following algorithmic steps are performed to advance the solution for a single iteration in the Second Order Godunov Method:

1. Local Extrapolation
2. Monotonicity Constraint
3. Characteristics Constraint
4. Riemann Problem Solution
5. Integration

It is interesting to note that most of these steps have an analog in conventional image processing methods. Here, we will give an explanation of the function of each algorithmic step of the Second Order Godunov Method and where applicable, will point to its possible analog in conventional signal processing techniques.

Step 1 consists of extrapolation of the values in the computational grid (pixel) cell to the edges of the cell. Linear or nonlinear extrapolation can be used. This step is analogous to the standard edge sharpening techniques used in image processing, with one important difference: the extrapolation is done not for the value itself but for its flux (change of value across cell boundary).

Step 2 includes a monotonicity constraint for the values at the cells' edges. This is analogous to the nonlinear technique of the locally monotonic regression[6] only recently introduced for signal processing.

Step 3 subjects the values at the edges to the constraints derived from a solution of one dimensional characteristics. This step assures that the values at the edges have not been extrapolated from directions inconsistent with the characteristic solutions. This prevents extrapolation as well as smearing or overshoot of the discontinuities. For the image processing application, this can be regarded as a form of automatic edge detection step where the shock waves are associated with the edges of an image.

Step 4 uses an exact solution of the system of the gas dynamic equations for calculation of the flux values based on the extrapolated values of the parameters at the left and right side of the edges. This step has no analogy in image processing. However, since the analytical solution includes discontinuities, an exact calculation of the flux at the edge location is allowed, even if this flux is calculated through a discontinuity.

Step 5 consists of finite volume integration of the system of conservation laws. Here, the image is effectively treated as a flow field; the flux integration serves as a smoothing filter from the image perspective.

Application of these steps can be considered as the application of a unique filter stack with proven properties of discontinuity preservation and robustness. Below we illustrate uses of this technique for practical problems of image processing that exemplify the feasibility and advantages of this approach.

The use of image analogies for image processing is not new. One widely applied technique treats an image as a potential field where the image potential acts as a force on the edges that are represented as elastic curves with some elastic properties.[3] Our approach, as stated, involves an application of a technique developed for gas dynamic problems for image deconvolution. Although this technique is very new, an analysis of the basic steps presented above and our experience with its application for image deconvolution show that this nonlinear algorithm has considerable potential for edge enhancement and filtering of extremely noisy signals.

3. IMAGE ENHANCEMENT BY THE SECOND ORDER GODUNOV METHOD

The field of gray scale intensity of an image can be translated into a flow field. To every image pixel we add a corresponding cell of the computational domain with values of the gas dynamical parameters proportional to the values of the gray scale. Since there are at least five gasdynamical parameters that can be defined in every cell of the computational domain (pressure, density, two velocity components and γ) and only one parameter in the image domain, cell mapping is not unique. Our understanding of the basic gasdynamical processes plays a major role in completing the analogy. Appropriate mapping of the image gray scale intensity into a flow field creates conditions favorable for the formation or enhancement of field discontinuities. For example, a shock wave reflecting from a wall or a contact surface can increase in strength, or two colliding flow streams will produce a contact surface that will become stronger in time. If we have a numerical technique to resolve these discontinuities accurately, then with successive numerical integration of the flow field, the discontinuities will sharpen as the solution evolves in time. Then by inverse mapping of the flow field to the image gray scale field, we can reconstruct an enhanced image. Below we give some examples of practical application of this technique.

3.1. Edge sharpening for a sinusoidal distribution

In Figure 2 results are given for edge definition of a one dimensional signal. The original sinusoidal signal is shown in Figure 2a. This example was chosen to test the ability of our technique to identify the edges of an image where the signal strength has deteriorated in the vicinity of the

edges, producing a gradual (instead of sharp) increase in the gray scale intensity. We observe that application of our technique results in significant sharpening of the edges, even after 15 or more iterations.

In Figure 3 random noise has been added to the sinusoidal signal shown in Figu 2a. The level of random noise addition corresponds to 10% of the maximum intensity of the ori al signal. The original signal with the random noise is shown in Figure 3a. In Figures 3b, 3c, 3d e observe successive noise filtering and edge enhancement with application of our algorithm fo: .5, 30, and 45 interations correspondingly. We see that the edges of the final processed signal : e located at exactly the same position as shown in Figure 2d for the uncontaminated signal.

Figure 4 illustrates the application of our algorithm to the signal that has been contaminated with 50% addition of random noise. Significant noise filtering occurs after 15 iterations and edge definition at the exact original locations after 45 iterations.

In Figure 5 the results are shown for a signal with 100% random noise added. Here again the signal is quickly filtered and the edges are picked up exactly at the correct locations.

3.2. Edge sharpening for a two dimensional image

Figure 3 contains a picture of Washington, DC taken from a Russian satellite. Digital representation of this picture had 150 dots per inch resolution. A fragment of the picture shown in Figure 6 is represented on an evenly spaced 400 × 360 grid. We take the gray scale pixel information of this picture and convert the data into initial conditions for a gasdynamic problem by assigning the values of pressure and density in the computational domain directly proportional to the values of the pixels on the gray scale. Now the gasdynamic problem is defined and we can solve it using our high resolution Second Order Godunov Method. In Figures 7a, 7b, and 7c results in the pixel plane are shown after three, six, and nine iterations respectively in the gasdynamic domain. By "iteration," we mean that the flow solver integration algorithm was applied to the given flow field, or in this case, the pressure and density data derived from the initial picture. Even after three iterations, the picture is significantly sharper and continues to improve with more iterations.

A more detailed examination of the sharpening effect can be obtained by looking at the one-dimensional cross section of the picture plane. In Figure 8, an arbitrary cross section of the original picture shown in Figure 6 is given. For clarity we show only the first fourth of the actual pixels in the cross section. We can see here that this particular cross section contains a multitude of sharp edges expressed only by three or four points. Further sharpening of these edges by a standard differentiation technique will lead to significant smearing of a number of the discontinuities. In Figure 9, the same cross section is shown after three iterations with the Second Order Godunov solver. Significant enhancement of all the sharp edges is evident. The process of enhancement can be followed in Figures 6b, 6c and 6d corresponding to six, nine, and twelve iterations. Continuous improvement in the definition of edges can be observed.

In Figures 10, 11a, 11b, 11c, and 11d, we demonstrate the ability of the current nonlinear PDE methodology to enhance simultaneously the high and low frequency features of an image. The amplitudes of both short and long wavelengths are simultaneously enhanced. However, as seen in the circled area, long wavelength features that retain one grid-cell discontinuities exhibit interesting behavior in that the cell-specific discontinuity, which appears in Figure 10, disappears in Figures 11b and 11c, but reappears in Figure 11d. The long wavelength definition continues to be enhanced in Figures 11a-11d. The origin of this behavior is presently unknown.

3.3. Application to Medical Imaging

Images of internal organs obtained with a Gamma Camera are usually of marginal quality and need significant post-processing to be useful for medical diagnostics. This is especially true if multiple pictures are taken of moving parts of the body, such as the heart, with low pixel resolution. In this section, we will demonstrate the application of our CFD technique for deconvolution of Gamma Camera images obtained during medical examinations.

Shown in Figure 12 is an image of the human heart produced by the staff of the Georgetown University Hospital, Department of Nuclear Medicine, using a Siemens Gamma Camera. This image contains a sequence of 64x64 pixel frames showing the heart at a sequence of time intervals. This plane image, originally recorded in 256 shades of gray scale, is presented here in 64 shades of gray. In Figures 12b, 12c and 12d the deconvoluted image is shown after 6, 12 and 18 processing iterations by our nonlinear technique. We observe in these figures a significant improvement in the image quality over the images in Figure 12a. Some of the diffuse edges in Figure 12a are clearly pronounced in Figures 12c and 12d. We have also applied our CFD technique to the Gamma Camera images of the brain and liver and have found a significant image deconvolution and edge enhancement.

4. CONCLUSIONS

The CFD technique described here for nonlinear signal and image processing is based on numerical techniques developed for Computational Fluid Dynamics, namely, the Second Order Godunov Method. We have demonstrated the application of this numerical method to signal processing, resulting in significant signal deconvolution and edge enhancement effects. Our preliminary analysis has shown that the Second Order Godunov Method, when applied to the gray scale intensity field of an image, is equivalent to an application of an unique filter stack. This filter stack has automatic edge detection, noise reduction and edge enhancement properties. We have demonstrated this nonconventional technique for the system of gas dynamic equations, where the Second Order Godunov Method assures high accuracy resolution of the flow discontinuities that are analogous to the edges in the image field. However, the same methodology can be applied to a reduced set of nonlinear hyperbolic partial differential equations, which will result in a significant optimization of the proposed technique.

5. REFERENCES

1. Nonlinear Image Processing, Proceedings of SPIE, Feb. 1990, Santa Clara, CA.
2. Osher, S. and L. Rudin, "Feature-Oriented Image Enhancement Using Shock Fielders," *Siam J. Numer. Anal.*, Vol. 27, pp. 919-940, NY, 1990.
3. Samadani, R. et al., "A Computer Vision System for Automatically Finding the Auroral Oval from Satellite Images," Image Processing Algorithms and Techniques. Proceedings of SPIE, 1244, 68-75, Feb. 1990.
4. Greene. R. R. et al., "Process and Apparatus for the Automatic Detection and Extraction of Features in Images and Displays," U.S. Patent 4,906,340, Mar. 1990.
5. Collela. P. and P. Woodward, "Piecewise Parabolic Method (PPM) for Gasdynamic Simulations." *J. Comp. Phys.*, Vol. 54, 174-201, 1984.
6. Restrepo, A. and A.C. Bovik, "Statistical Optimality of Locally Monotonic Regression." Nonlinear Image Processing, Proceedings of SPIE, 1247, 89-99, Feb. 1990.

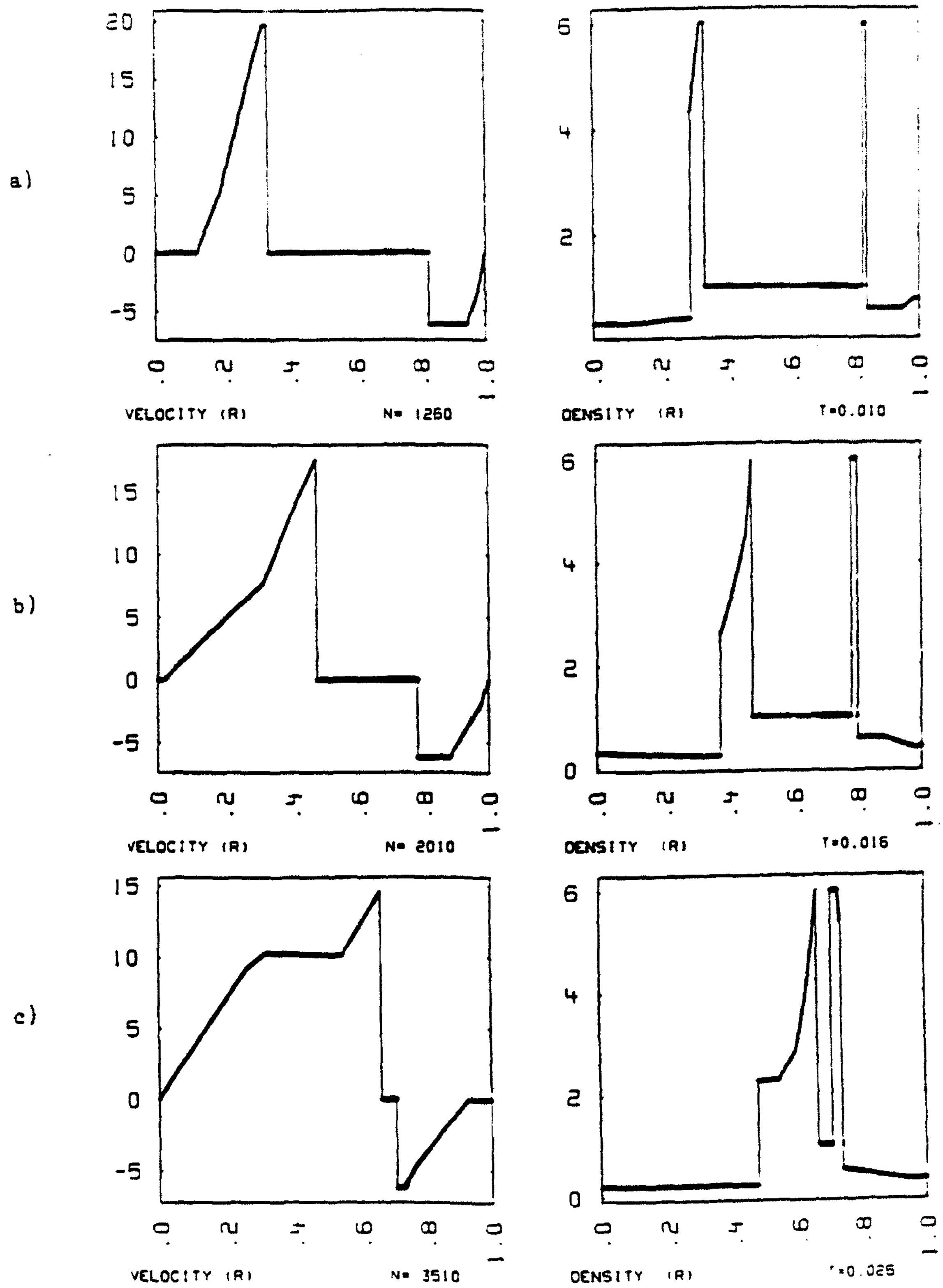


Fig. 1. High resolution of flow discontinuities obtained with the Second Order Godunov Method.

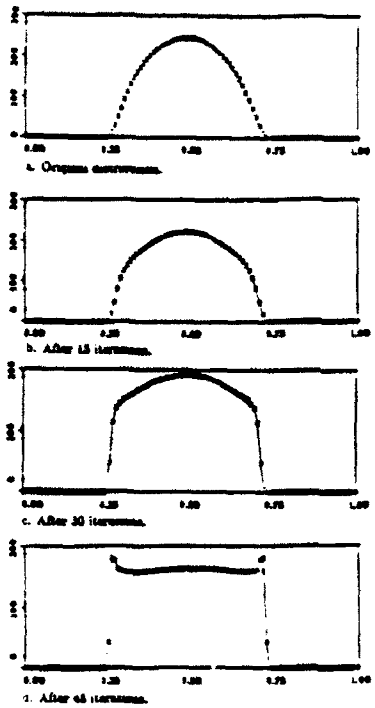


Fig. 2. Edge enhancement for a sinusoidal distribution without noise.

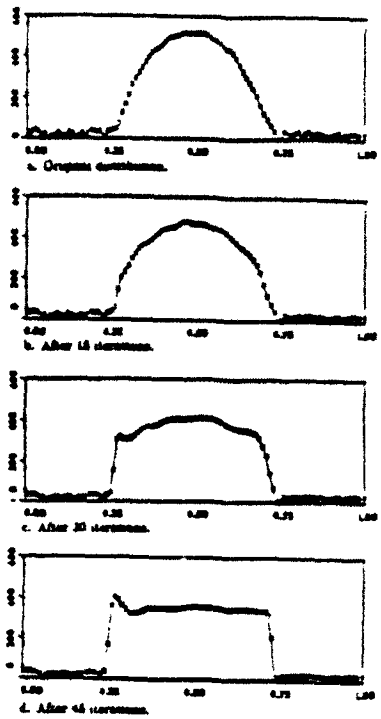


Fig. 3. Edge enhancement for a sinusoidal distribution with 10% intensity random noise.

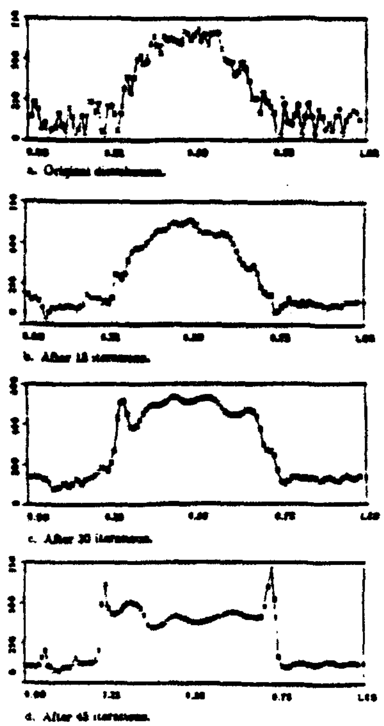


Fig. 4 Edge enhancement for a sinusoidal distribution with 50% intensity random noise.

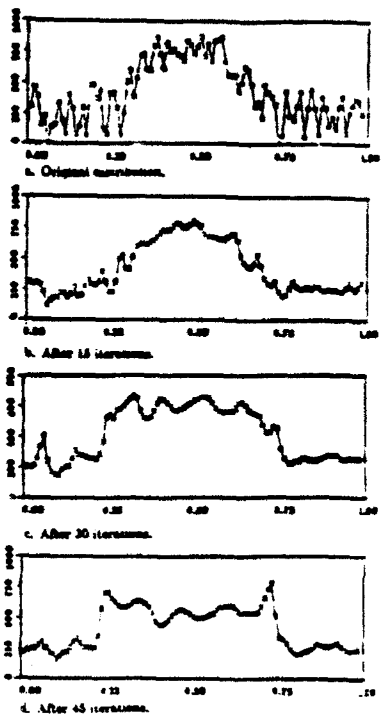


Fig. 5. Edge enhancement for a sinusoidal distribution with 100% intensity random noise.

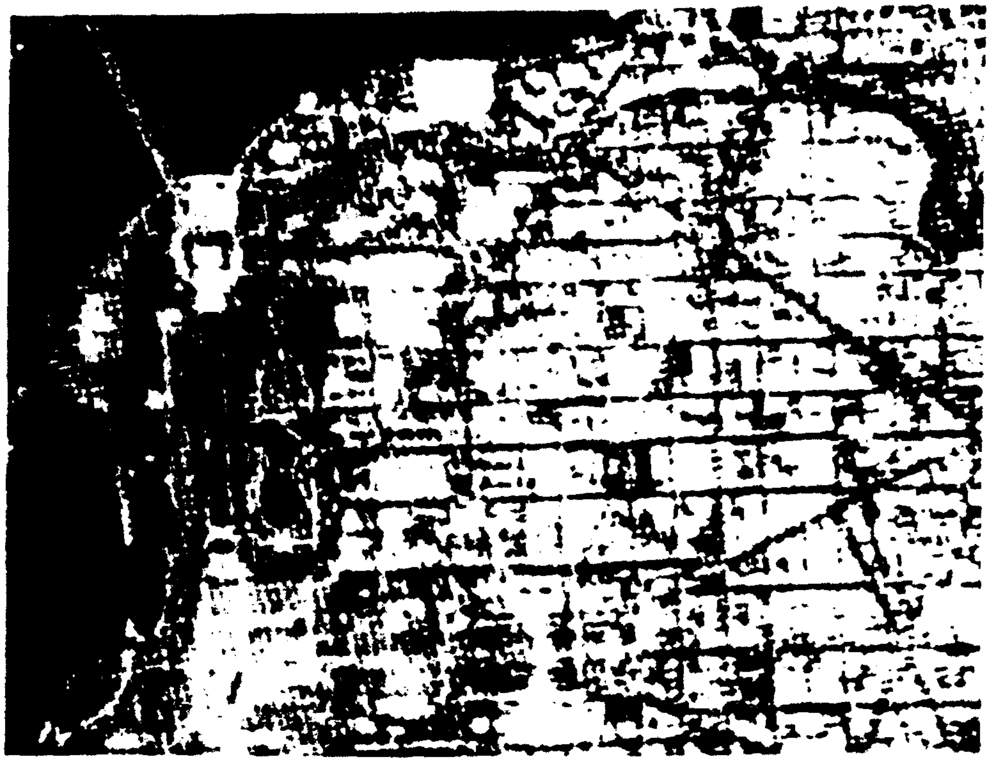


Fig. 6. The original satelite photograph of Washington. DC with resolution reduced to 150 dots/inch.

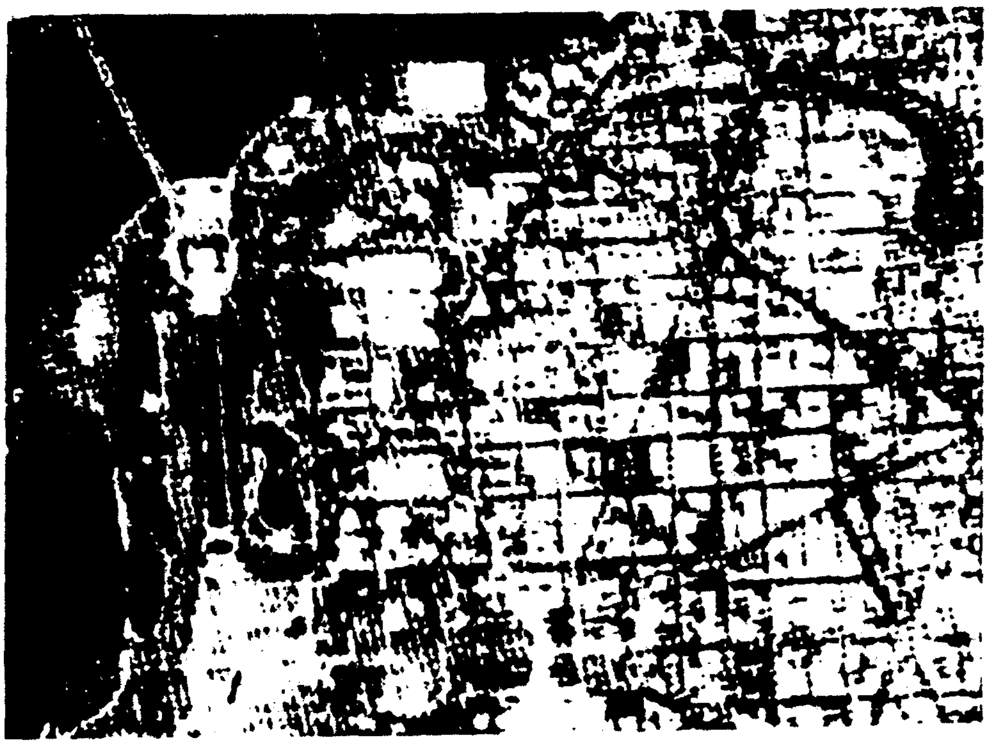


Fig. 7a. The sharpened picture after the Godunov solver has been used. After three iterations. Note the details that appeared on the Potomac. These details are barely visible even on the original high resolution photograph.

Fig. 7b. After six iterations.

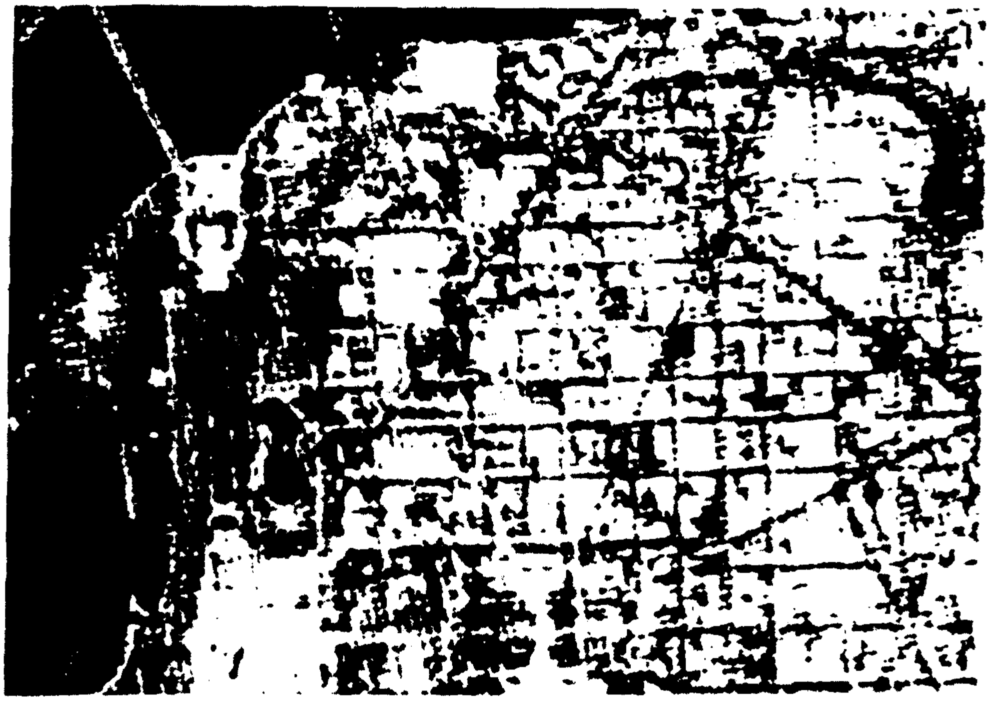


Fig. 7c. After nine iterations.

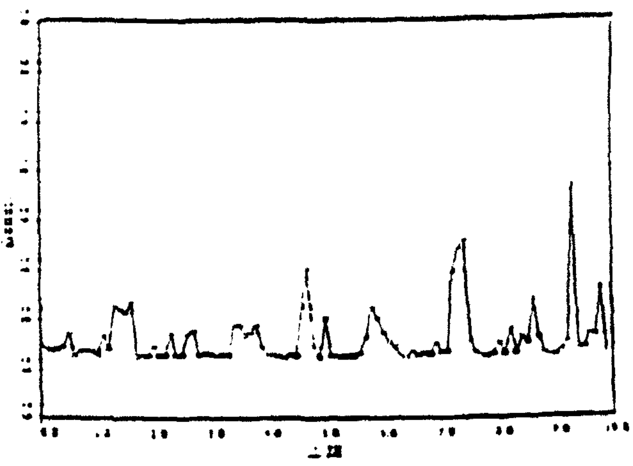


Fig. 8. Gray scale density of a cross section of the original image.

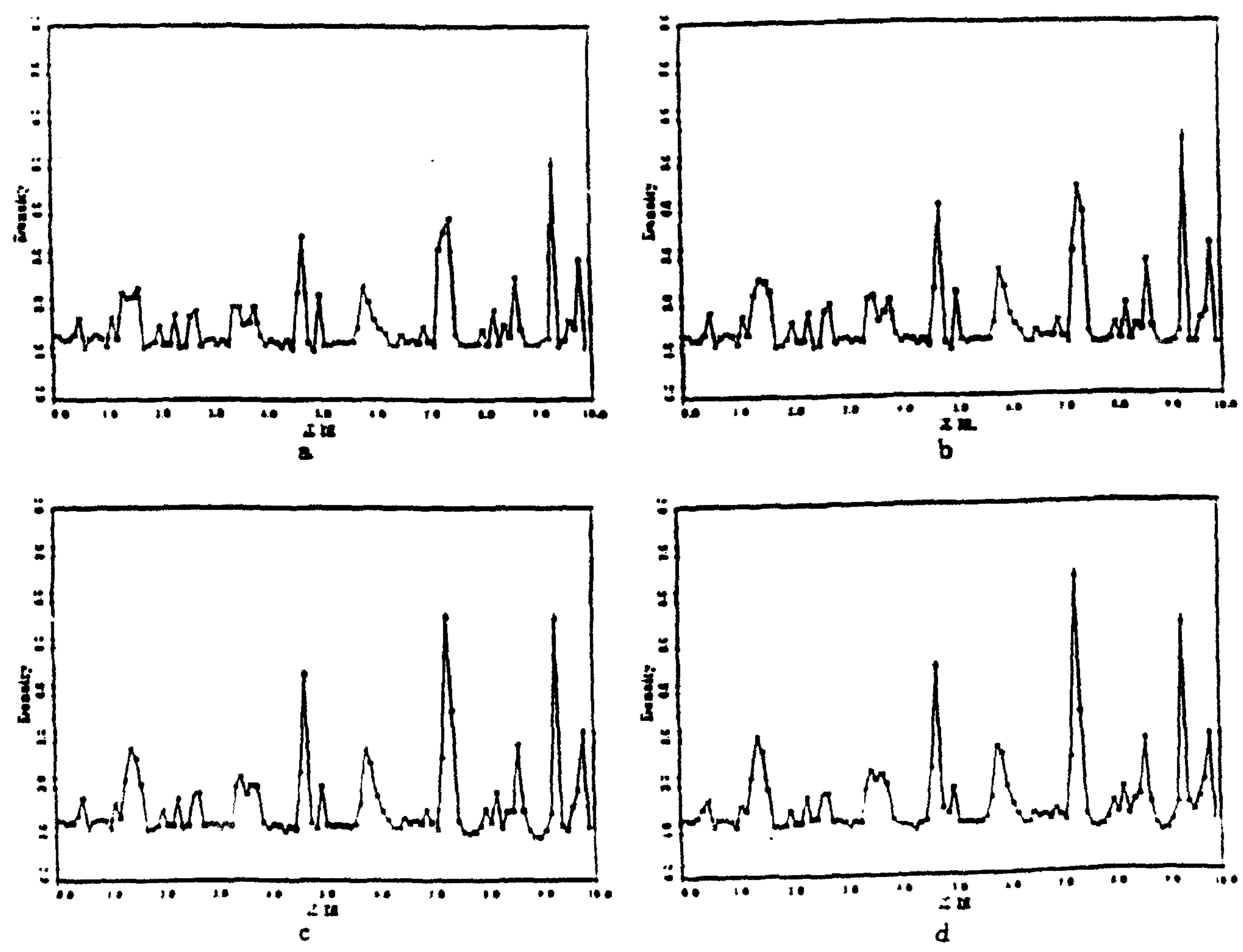


Fig. 9. Gray scale density of the CFD processed image: (a) after 3 iterations; (b) after 6 iterations; (c) after 9 iterations; (d) after 12 iterations.

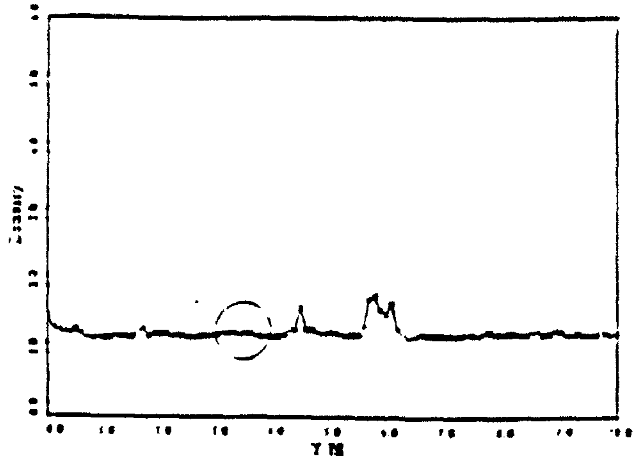


Fig. 10. Gray scale density of a cross section of the original image.

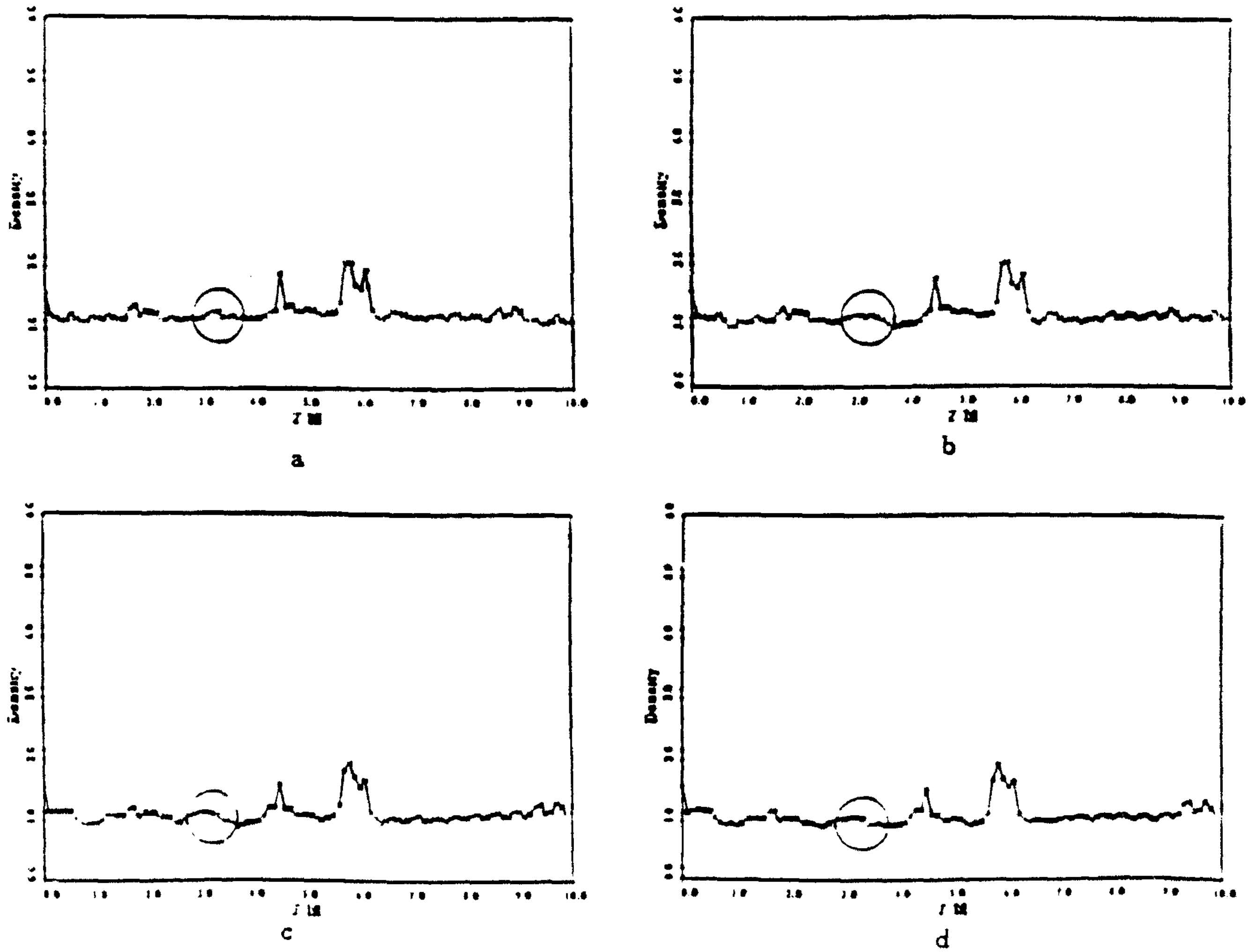


Fig. 11. Gray scale density of the CFD processed image: (a) after 3 iterations; (b) after 6 iterations; (c) after 9 iterations; (d) after 12 iterations.

(a) (b) (c) (d)

Fig.12. Image of human heart taken by a Siemens Gamma Camera. (a) Original image 64x64 pixels per frame; (b) Image after six processing iterations; (c) Image after 12 processing iterations; (d) Image after 18 processing iterations.

Review of Propulsion Applications and Numerical Simulations of the Pulsed Detonation Engine Concept

S. Eidelman,* W. Grossmann,† and I. Lottati*
Science Applications International Corporation, McLean, Virginia 22102

Here we review experimental and computational studies of the pulsed detonation engine concept (PDEC) and present results of our recent numerical study of this concept. The PDEC was proposed in the early 1940s for small engine applications; however, its potential was never realized due to a complicated, unsteady operation regime. In this study, we demonstrate the use of current advances in numerical simulation for the analysis of the PDEC. The high-thrust/engine volume ratio obtained in our simulations demonstrates promising potential of the pulsed detonation engine concept.

Introduction

EARLY developments of engine technology leading to practical propulsion engines were almost completely associated with steady-state engine concepts. Unsteady concepts, which initially appeared promising, never evolved from the conceptual state and have remained for the most part unexplored. The early work in unsteady propulsion suffered from a lack of appropriate analytical and design tools, a condition which seriously impeded the advancement of the unsteady concepts to a practical stage.

In this paper, we review the historical development of unsteady propulsion by concentrating on the particular concept of the intermittent detonation engine, and discuss current research activities in this area. A review of the literature[1-24] reveals that a significant body of experimental and theoretical research exists in the area of unsteady propulsion. However, this research has not been extended to the point where a conclusive quantitative comparison can be made between impulsive engine concepts and steady-state concepts. For example, the analysis given in Refs. 8-11 of the performance of a detonation engine concept includes neither frequency dependence nor analysis of losses due to multicycle operation. A new generation of analytical and computational tools exists today and allows us to revisit and analyze such issues with a high degree of confidence. Numerical simulation has developed to the state where it can now provide time-dependent two- and three-dimensional modeling of complex internal flow processes[20,24,25] and will eventually result in tools for systematically analyzing and optimizing engineering design. In addition to a review of applications of the pulsed detonation engine concept (PDEC), we will report results of a numerical study of an air-breathing detonation engine. This study was performed using new unsteady computational fluid dynamics (CFD) tools that we will also describe.

Our paper is structured as follows: 1) historical review of the pulsed detonation development efforts; 2) description of the basic phenomenology of the air-breathing pulsed detonation engine concept; 3) description of the mathematical formulation and new numerical scheme used to simulate the problem; 4) discussion of the simulation results; and 5) conclusions.

Presented as Paper 89-2446 at the AIAA/ASME/ASEE/SAE 25th Joint Propulsion Conference, Monterey, CA, July 10-12, 1989; received Sept. 8, 1989; revision received April 12, 1990. This paper is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

*Senior Scientist, Applied Physics Operation. Member AIAA.
†Chief Scientist, Applied Physics Operation. Member AIAA.

Historical Review

Constant-Volume Combustion

From the very early development of jet-propulsion engines, it was known that an engine based on a constant-volume combustion process achieves higher thermodynamics efficiency than a constant pressure engine. This follows from a thermodynamic analysis of the engine cycle.[1]

Constant-volume combustion was used in gas turbine engines at the beginning of this century, and the first gas turbine engines in commercial use were based on the constant-volume cycle. Jet-propulsion engines were one of the applications of the constant volume cycle (or explosion cycle) which was explored in the late 1940s.[2] Although the explosion cycle operates at a larger pressure variation in the combustion chamber than in a pulse jet,[3,4] the cycle actually realized in these engines was not a fully constant-volume one since the combustion chamber was open-ended.[2] In Ref. 2, the maximum pressure ratio measured in an explosion cycle engine was 3:1, whereas the pressure ratio for the same mixture under the assumption of a constant-volume cycle would be 8:1. Also, this engine was limited by the available frequency of cycles, which in turn was limited by the reaction rate. A simple calculation[2] showed that if the combustion time could be reduced in this engine from 0.006–0.003 s, the thrust per pound of mixture would increase 100%. Thus, the explosion-cycle engine has two main disadvantages:

1) Constrained volume combustion (as distinguished from constant-volume combustion) does not take full advantage of the pressure rise characteristic of the constant-volume combustion process.

2) The frequency of the explosion cycle is limited by the reaction rate, which is only slightly higher than the deflagrative combustion rate.

The main advantage of the constant-pressure cycle is that it leads to engine configurations with the steady-state processes of injection of the fuel and oxidizer, combustion of the mixture, and expansion of the combustion products. These stages can be easily identified and the engine designer can optimize them on the basis of relatively simple steady-state considerations.

At the same time, an engine based on constant-volume combustion will have an intermittent mode of operation, which may complicate its design and optimization. We are interested in the question of whether this complication is worth the potential gains in engine efficiency.

Pulsed Detonation Engine as an Ultimate Constant-Volume Combustion Concept

The detonation process, due to the very high rate of reaction, permits construction of a propulsion engine in which the

constant-volume process can be fully realized. In detonative combustion, the strong shock wave, which is part of the detonation wave, acts like a valve between the detonation products and the fresh charge. The speed of the detonation wave is about two orders of magnitude higher than the speed of a typical deflagration. This allows the design of propulsion engines with a very high power density. Usually, each detonation is initiated separately by a fully controlled ignition device, and the cycle frequency can be changed over a wide range of values. There is only an upper limit for the detonation cycle frequency. This limit is determined by the time it takes to refill the detonation chamber with the fresh combustible mixture. This in turn will depend on chamber geometry and the external flow parameters. In our study, we have established that detonation frequencies of 200–250 Hz appear to be feasible. At the same time, the same PDEC engine can operate at very low detonation frequency with thrust almost linearly proportional to the frequency. This also means that a device based on a detonative combustion cycle can be scaled, and its operating parameters can be modified for a range of required output conditions. There have been numerous attempts to take advantage of detonative combustion for engine applications. In the following, we give a description of the most relevant past experimental and analytical studies of the detonation engine concept.

Hoffmann's Report

The first reported work on intermittent detonation is attributed to Hoffmann[5] in 1940. He operated an intermittent detonation test stand with acetylene-oxygen and benzine-oxygen mixtures. The addition of water vapor was used to prevent the highly sensitive acetylene-oxygen mixture from premature detonation. Hoffmann[5] indicated the importance of the spark plug location in reference to tube length and diffuser length. It was found that a continuous injection of the combustible mixture leads to only a narrow range of ignition frequencies that will produce an intermittent detonation cycle. These frequencies are governed by the time required for the mixture to reach the igniter, the time of transition from deflagration to detonation, and the time of expansion of the detonation products. Hoffmann attempted to find the optimum cycle frequency experimentally. It was discovered that detonation-tube firing occurred at lower frequencies than the spark-plug energizing frequencies, indicating that the injection flow rate and ignition were out of phase. World War II prevented further work by Hoffmann and co-workers.

Nicholls' Experiments

A substantial effort in intermittent detonation engine research was done by a group headed by Nicholls[6-10] of the University of Michigan beginning in the early 1950s. The most relevant work concerns a set of experiments carried out in a 6-ft-long detonation tube.[6] The schematics of the detonation-tube experiments test rig used by Nicholls and co-workers are shown in Fig. 1. The detonation tube was contructed from a 1-in.-i.d. stainless-steel tube. The fuel and oxidizer were injected under pressure from the left end of the tube and ignited at the 10-in. distance downstream. The tube was mounted on a pendulum platform that was suspended by support wires. Thrust for single detonations was measured by detecting tube (platform) movement relative to a stationary pointer. For multicycle detonations, thrust measurement was achieved by mounting the thrust end of the tube to the free end of the cantilever beam. In addition to direct thrust measurements, the temperature on the inner wall of the detonation tube was measured.

Fuel mixtures of hydrogen/oxygen, hydrogen/air, acetylene-oxygen, and acetylene-air were used. The gaseous oxidizer and fuel were continuously injected at the closed end wall of the detonation tube and three fixed flow rates were used. Under these conditions, the only parameters that could be varied were the fuel/oxidizer ratio and frequency of ignition. A maximum gross thrust of $\approx$ 3.2 lb was measured in hydrogen/air mixture at the frequency of $\approx$ 30 detonations/s. The most promising results were demonstrated for the hydrogen/air mixture, where a fuel specific impulse of $I_{sp} = 2100$ s was reached. The maximum frequency of detonations obtained in all experiments was 35 Hz. The temperature measurements on the inner wall showed that for the highest frequency of detonations the temperature did not exceed 800°F.

In their later work,[8-10] the University of Michigan group concentrated on development of the rotating detonation wave rocket motor. No further work on the pulsed detonation cycle was pursued.

Krzycki's Experiments

In a setup somewhat similar to Nicholls', Krzycki[11] performed an experimental investigation of intermittent detonations with frequencies up to 60 cps. An attempt was also made to analyze the basic phenomena using unsteady gas dynamic theory. Krzycki's attempt to analyze the basic phenomena relied on wave diagrams to trace characteristics, assumptions of isentropic flow for detonation and expansion, and incompressible flow for mixture injection processes. The most convincing data from the experiments are the measurement of thrust for a range of initiation frequencies and mixture flow rates. Unfortunately, no direct pressure measurement in the device are reported so that only indirect evidence exists of the nature of the process observed.

The basic test stand used by Krzycki is very similar to that used by Nicholls et al.[6] The length of the detonation tube and internal diameter were exactly the same as those in Nicholls' experiments. A propane/air mixture was continuously injected through reversed-flow diffuser for better mixing and

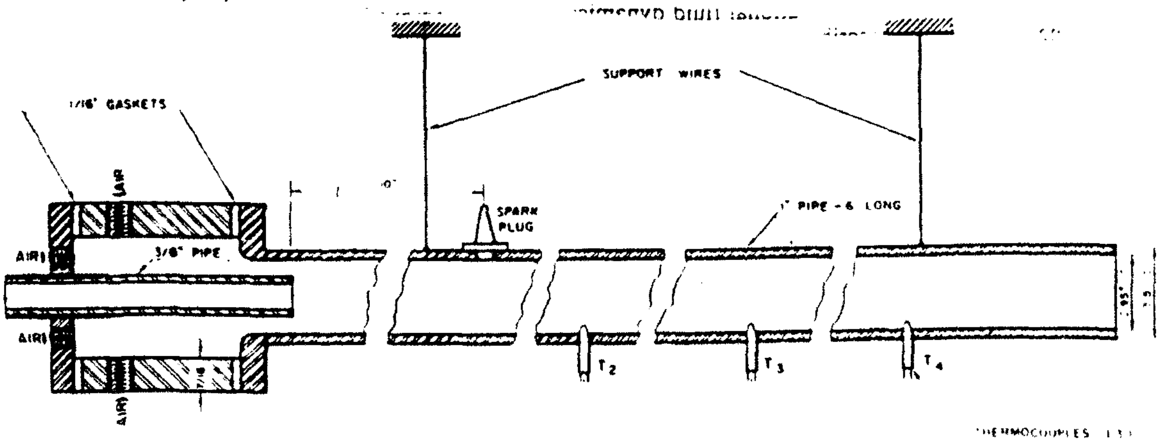


Fig. 1 Detonation tube used in experiments by Nicholls et al.

ignited at the 25-cm distance from the injection point by an automobile spark plug. The spark frequency was varied from 1–60 Hz. The spark plug power output was varied inversely with the initiation frequency and at the frequency of 60 Hz was only 0.65 J. This fact alone eliminated the possibility of direct initiation of the detonation wave by the spark and consequently all of the experiments were performed in the region dominated by transition from deflagration to detonation. According to experimental data and theory,[12] for direct initiation of a mixture of propane/air at the detonability limits, an energy release on the order of 10^6 J is required. Thus, the required deflagration-detonation transition region length would have been prohibitively large for the propane/air mixture. It follows that in all of the experiments a substantial part of the process was deflagrative. This resulted in low efficiency and negligible thrust. Krzycki repeated the experiments of Nicholls using exactly the same size detonation tube and basically the same rates of injection of the detonatable mixture. Krzycki's experimental results are very well-documented, giving enough information to deduce a clear picture of the physical processes occurring in the tube. A conclusion, arrived at by the author, was that thrust was possible from such a device but practical applications did not appear promising. It is unfortunate that, possibly based on Krzycki's extensive but misleading results, all experimental work related to the pulsed detonation engine concept stopped at this time.

Work Reported in Russian Sources on Pulse Detonation Devices

A review of the Russian literature has not uncovered work concerning applications of pulsed detonation devices to propulsion. However, there are numerous reports of applications of such devices for producing nitrogen oxide[13] (an idea proposed in the 1940s by Zeldovich to use detonation for binding nitrogen directly from air to produce fertilizers) and as rock crushing devices.[14]

Korovin et al.[13] provide a most interesting account of the operation of a commercial detonation reactor. The main objective of this study was to examine the efficiency of thermal oxidation of nitrogen in an intermittent detonative process as well as an assessment of such technological issues as the fatigue of the reactor parts exposed to the intermittent detonation waves over a prolonged time. The reactor consisted of a tube with an inner diameter of 16 mm and length 1.3 m joined by a conical diffuser to a second tube with an inner diameter of 70 mm and length 3 m. The entire detonation reactor was submerged in running water. The detonation mixture was introduced at the end wall of the small tube. Methane, oxygen, and nitrogen comprised the mixture composition and the mixture ratios were varied during the continuous operation of the reactor. The detonation wave velocity was measured directly by piezoelectric sensors placed in the small and large tubes. The detonation initiation frequency in the reactor was 2–16 Hz. It is reported that the apparatus operated without significant changes for 2000 h.

Smirnov and Boichenko[14] studied intermittent detonations of a gasoline/air mixtures in a 3-m-long and 22-mm-i.d. tube operating in the 6–8 Hz ignition frequency range. The main motivation of this work was to improve the efficiency of a commercial rock-crushing apparatus based on intermittent detonations of the gasoline/air mixtures.[15] The authors investigated the dependence of the length of the transitional region from deflagration to detonation on the initial temperature of the mixture.

As a result of the information contained in the Soviet reports, it can be concluded that reliable commercial devices based on intermittent detonations can be constructed and operated.

Development of the Blast Propulsion System at JPL

Back,[16] Varsi et al.,[17] Kim et al.,[18] and Back et al.[19] at the Jet Propulsion Laboratory (JPL) studied the feasibility of a rocket thruster powered by intermittent detonations of solid explosive. The main application foreseen by the authors is propulsion in dense or high-pressure atmospheres of certain solar system planets. The JPL work was directed at very specific applications; however, the studies[17-19] addressed some key issues of devices using unsteady processes such as propulsion efficiency. The JPL studies have important implications to pulsed detonation propulsion systems.

Reference 19 gives the basic description of the test stand used. In this work, a data sheet type C explosive was detonated inside a small detonation chamber attached to nozzles of various length and geometry. The nozzles, complete with firing plug, were mounted in a containment vessel that could be pressurized with the mixture of various inert gases from vacuum to 70 atm. The apparatus measured directly the thrust generated by single detonations of a small amount of solid explosive charge expanding into conical or straight nozzles. Thrust and specific impulse were measured by a pendulum balance system.

Results obtained from an extensive experimental study of the explosively driven rocket have led to the following conclusions. First, rockets with long nozzles show increasing specific impulse with increasing ambient pressure in carbon dioxide and nitrogen. Short nozzles, on the other hand, show that specific impulse is independent of ambient pressure. Most importantly, most of the experiments obtained a relatively high specific impulse of 250 s and larger. This result is all the more striking since the detonation of a solid explosive yields a relatively low energy release of approximately 1000 cal/g compared with 3000 cal/g obtained in hydrogen/oxygen combustion. Thus, it can be concluded that the total losses in a thruster based on unsteady expansion are not prohibitive and, in principle, very efficient propulsion systems operating on intermittent detonations are possible.

Detonation Engine Studies at Naval Postgraduate School

A modest exploratory study of a propulsion device utilizing detonation phenomena was conducted at the Naval Postgraduate School (NPS).[20-23] During this study, several fundamentally new elements were introduced to the concept distinguishing the new device from previous ones.

First, it is important to note that the experimental apparatus constructed by Helman et al.[22] showed the first successful self-aspirating, air-breathing detonation device. Intermittent detonation frequencies of 25 Hz were obtained. This frequency was in phase with the fuel-mixture injection through timed fuel-valve opening and spark discharge. The feasibility of intermittent injection was established. Pressure measurements showed conclusively that a detonation process occurred at the frequency chosen for fuel injection. Furthermore, self-aspiration was shown to be effective. Finally, the effectiveness of a primary detonation as a driver for the main detonation was clearly demonstrated. Although the NPS studies were abbreviated, many of the technical issues considered to be essential for efficient intermittent detonation propulsion were addressed with positive results.

Simulations of Pulsed Detonation Engine Cycle at NASA Ames Research Center

Recently Cambier and Adelman[24] carried out numerical simulations of a pulsed detonation engine cycle taking into account finite-rate chemistry. Unfortunately, the simulations were restricted to a quasi-one-dimensional model. The configuration considered had a 6-cm-i.d., 50-cm-long main chamber that was attached to a 43-cm-long diverging nozzle. It was assumed that a stoichiometric mixture of hydrogen/air at 3.0 atm is injected from an inlet on the closed end wall of the detonation chamber. Under these conditions, Cambier and Adelman estimated a large range of possible detonation frequencies of engine operation up to 667 Hz. The origin of this estimate is not clear from their work since, according to their simulations, the detonation, expansion, and fresh charge fill requires 2.5 ms. This value leads to a maximum frequency of

400 Hz. The simulated engine performance yielded a large average thrust of ≈ 4000 N and an unusually high specific impulse of 6507 s. These simulations were the first to demonstrate the use of modern CFD methods to address the technical issues associated with unsteady pulsed detonation concepts.

In the remaining sections, we discuss a particular propulsion concept based on the results of the experiments of Helman et al.[22] and describe a computational study of its performance characteristics. The unsteady numerical scheme used for the study made use of unique simulation techniques; the key ingredients of these techniques are also described.

Generic Pulsed Detonation Engine

The generic device we consider here is a small cylindrical engine, 15 cm long and 15 cm in diameter. The combustible gas mixture is injected at the closed end of the detonation chamber and a detonation wave propagates through the mixture. The size of the engine suggests a small payload, but the concept can be extended to larger payloads simply by scaling up the size of the detonation chamber and possibly combining a number of engines into one large propulsion engine. A key issue in the pulsed detonation engine concept is the design of the main detonation chamber. The detonation chamber geometry determines the propulsion efficiency and the duration of the cycle (frequency of detonations). Since the fresh charge for the generic engine is supplied from the external flowfield, the efficiency of the engine depends on the interaction of the surrounding flow with the internal flow dynamics. The range of the physical processes requiring simulation in order to model the complex flow phenomena associated with the detonation engine performance is very broad. A partial list is as follows:

1) Initiation and propagation of the detonation wave inside the chamber.

2) Expansion of the detonation products from the chamber into the airstream around the chamber at flight Mach numbers.

3) Reverse flow from the surrounding air into chamber resulting from overexpansion of the detonation products.

4) Pressure buildup in the chamber due to reverse flow. The flow pattern inside the chamber during postexhaust pressure buildup determines the strategy for mixing the next detonation charge.

5) Strong mutual interaction between the flow processes inside the chamber and flow around the engine.

All of these processes are interdependent and their timing is crucial to the engine efficiency. Thus, unlike simulations of steady-state engines, the phenomena described above cannot be evaluated independently.

The need to resolve the flow inside the chamber accounting for nozzles, air inlets, etc., and at the same time resolve the flow around the engine, where the flow regime varies from high subsonic, locally transonic, and supersonic, makes it a challenging computational problem.

The main issue is to determine the timing of the air intake for the fresh gas charge. It is sufficient to assume inviscid flow for the purpose of simulating the expansion of the detonation products and fresh gas intake. In the following, we present the first results of an inviscid simulation of the detonation cycle in a cylindrical chamber. First, we describe our computational method for solving the time-dependent Euler equations used in the study.

Unsteady Euler Solver

A new second-order algorithm for solving the Euler equations on an unstructured grid was used in our study of the detonation concept. The approach is based on first- and second-order Godunov methods. The method leads to an extremely efficient and fast flow solver that is fully vectorized and easily lends itself to parallelization. The low memory requirements and speed of the method are due to the use of a unique data structure.

Until recently most CFD simulations were carried out with logically structured grids. Vectorization and/or parallelization did not present a problem. The increased need for simulation of flow phenomena in the vicinity of complex geometrical bodies and surfaces has led to the development of CFD codes for logically unstructured grids. The most successful of these unstructured grid codes are based on finite elements or finite volume methods. For an unstructured grid in two dimensions, the computational domain is usually covered by triangles, and the indices of the arrays containing the values of the hydrodynamic flow quantities are not related directly to the actual geometric location of a node. The calculations performed on unstructured grids evolve around the elemental grid shape (e.g., the triangle for two-dimensional problems), and there is no obvious pattern to the order in which the local integrations should be performed. Explicit integration of hydrodynamic problems on an unstructured grid requires that a logical substructure should be created which identifies the locations in the global arrays of all of the local quantities necessary for the integration of one element. This usually results in a large price in computational efficiency, in memory requirements, and in code complexity. As a consequence, vectorization for the conventional unstructured grid methods has concentrated on rearrangement of the data structure in a manner such that these locally centered data structures appear as global arrays. This can be done to some extent using machine dependent gather-scatter operations.[25,26] Additional optimization can be achieved using localization and search algorithms. However, these methods are complex and result in marginal improvement. Most optimized unstructured codes to date run considerably slower and require an order of magnitude more memory per grid cell than their structured counterparts. Parallelization of the conventional unstructured codes is even more difficult, and there is very little experience with unstructured codes on massively parallel computers.

The method we have developed overcomes these difficulties and results in codes with speed and memory requirements comparable to those found in structured grid codes. Moreover, the ability to construct grids with arbitrary resolution leads to a flexibility in dealing with complex geometries not attainable with structured grids. The essence of the method is based on an independent flux calculation across the edges of a dual baricentric grid, followed by node integration. This approach is order independent. Below we give the essential details of our algorithm; a complete description follows later.

Basic Integration Algorithm

We begin by describing the first-order Godunov method for the system of two-dimensional (axisymmetric) Euler equations written in conservation law form as

$$\frac{\partial Q}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial r} = -\frac{1}{r}C \tag{1}$$

where

$$Q = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ e \end{pmatrix}, \quad F = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ (e+p)u \end{pmatrix}$$

$$G = \begin{pmatrix} \rho v \\ \rho v u \\ \rho v^2 + p \\ (e+p)v \end{pmatrix}, \quad C = \begin{pmatrix} \rho v \\ \rho v u \\ \rho v^2 \\ (e+p)v \end{pmatrix}$$

Here u and v are the x and r velocity vector components, p the pressure, ρ the density, and e the total energy of the fluid per unit volume. It is assumed that a mixed (initial conditions, boundary conditions) problem is properly posed for the set,

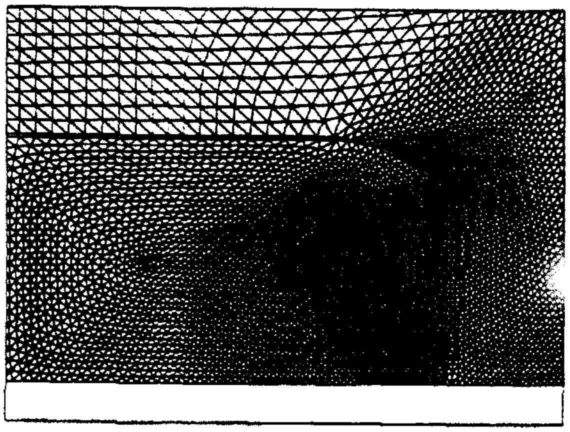


Fig. 2a Computational domain and grid used in simulation of PDEC operation.

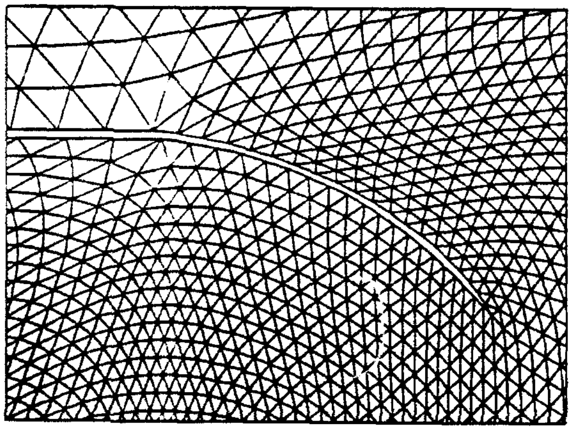


Fig. 2b Enlargement of computational grid in the vicinity of the PDEC nozzle.

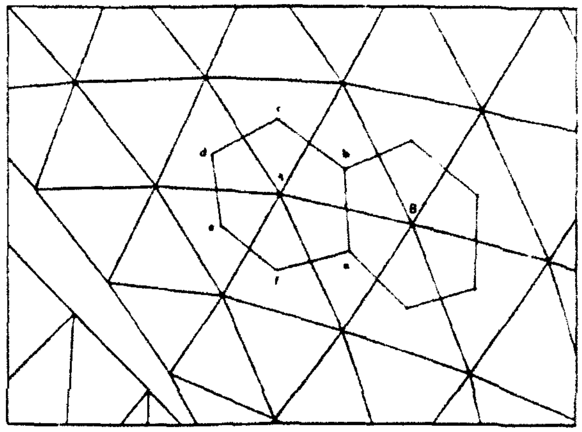


Fig. 3 The primary (triangles) and secondary (polygons) unstructured grids.

Eq. (1), and that an initial distribution of the fluid parameters is given at $t = 0$ and some boundary conditions defining a unique solution are specified on the boundary of the computational domain.

We look for a solution of the system of equations represented by Eq. (2) in the computational domain covered by an unstructured grid. As an example, Fig. 2a shows the unstructured triangular grid used in the pulsed detonation engine simulation. Here most of the computational effort is committed to the resolution of the flow inside the engine detonation chamber and in the immediate vicinity of the nozzle. In Fig. 2b, an enlargement of the nozzle region is shown, illustrating the ability to represent geometry of arbitrary complexity and with localized resolution.

Figure 3 displays a fragment of the computational domain with the corresponding dual grid. The secondary or dual grid is formed by connecting the baricenters of the primary mesh, thus forming finite polygons around the primary vertices.

We have found, as have others,[27] that the best practical representation of the integration volume is obtained when the dual grid is formed by connecting baricenters of the triangles. Integration by the Godunov method[28] can be divided into two basic steps: 1) calculation of the fluxes at the edges of the secondary grid using solutions of a set of one-dimensional Riemann problems; and 2) integration of the system of partial differential equations, which amounts to addition of all of the fluxes for every polygon at a particular time step.

To define the fluxes for the grid shown in Fig. 3 at every edge of the main grid, it is necessary to solve the corresponding Riemann problem. For example, to define the flux at the edge *ab*, we solve the Riemann problem between points A and B. The solution of this problem is in coordinates local to the edge of the dual grid *ab* so that the tangential component of velocity will be directed along this edge (*ab*). Implementation of our approach requires maintaining strict consistency when defining the "left" and "right" states for the Riemann problems at the edges *ab*, *bc*, *cd*, *de*, *ef*, and *fa*. For this reason, we define not only the location of the vertices and lengths of the edges but also the direction of the edges with respect to the primary grid. For the clockwise integration pattern in the same polygon, point A will be the "right" state for all of the Riemann problems related to this point, and the neighbor will represent the "left" side of the diaphragm.

It is easy to see that the flux calculation is based on information at only two nodes and requires single geometrical parameters defining the edge of the secondary grid that dissects the line connecting the two points. Thus, we can calculate all of the values needed for flux calculation in one loop over all edges of the primary grid without any details related to the geometrical structures that these edges form. This in turn assures parallelization or vectorization of the algorithm for the bulk of the calculations involving the Riemann solver that provides the first-order flux. The only procedure not readily parallelizable is the integration of the fluxes for the flow variables at the vertices of the grid. Here we use the "edge coloring" technique that allows us to split the flux addition loop into seven or eight loops for edges of different color. Each of these loops is usually large enough not to impair vectorization. At this stage, all of the fluxes are added with their correct sign corresponding to the chosen direction of integration within the cell. The amount of calculation required here is minimal since the fluxes are known and need only to be multiplied at each time step by a simple factor and added to the vertex quantity.

Second-Order Integration Algorithm

The second-order solver is constructed along lines similar to that of the first-order method. At each cell edge, the Riemann problem is solved for some specified pair of left and right conditions. The solution to this Riemann problem is then used in the calculation of fluxes that are added later to advance to the next integration step. The extension to second-order is achieved by using extrapolation in space and time to obtain time-centered left and right-limiting values as inputs for the Riemann problem. The basic implementation of the method of calculation of second-order accurate fluxes is fundamentally the same as for one-dimensional cases. The only difference is in the method of obtaining linear extrapolation of the flow variables as a first guess of their value at the edges of the dual grid. To obtain the first guess, we need to know the gradient of some gasdynamical parameter U at the vertices of the primary mesh. The value of ∇U can be evaluated by using

a linear path integral along the edges, which delineates the finite volume associated with the vertex. For vertex A in Fig. 3,

$$\int_A \nabla U \mathrm{d}A = \oint_l Un \, \mathrm{d}l \tag{2}$$

where integration along the path l in this case is equivalent to integration along the edges ab, bc, cd, de, ef, and fa. Knowing the gradient of the gasdynamic parameter in the volume related to vertex A will allow us to extrapolate the values of this parameter at any location within the volume. This permits us to evaluate the first guess for U at the edges of the dual grid. The final implementation of the second-order algorithm has been described previously.

A schematic flowchart of the basic steps of the second-order algorithm implementation is shown in Fig. 4.

Simulations of the Generic Pulsed Detonation Engine

In this section, we present sample results of simulations of the generic PDE device using the numerical code described in the preceding section. In Fig. 2a, the computational domain containing the PDE main detonation chamber is shown covered with the unstructured grid. In our sample simulation, we have chosen a small ≈ 15-cm-long and ≈ 15-cm-i.d. cylindrical chamber with a small converging nozzle. This geometry is one of a number of the geometries we have analyzed in a parametric study whose goal was to evaluate and optimize a typical PDEC device. The device shown in Fig. 1a does not represent the optimum and is given here to illustrate our methodology. We consider a situation when the PDEC serves as a main thruster for a vehicle traveling in air with the velocity of $M = 0.9$ and located at the aft end of the vehicle. The main objectives of the simulations presented here are as follows:

1) To find the maximum cycle frequency. This is determined by the time required from detonation, exhaust of combustion products, and intake of fresh charge for the next detonation.

2) To calculate the thrust produced during each cycle and the integrated thrust as a function of time.

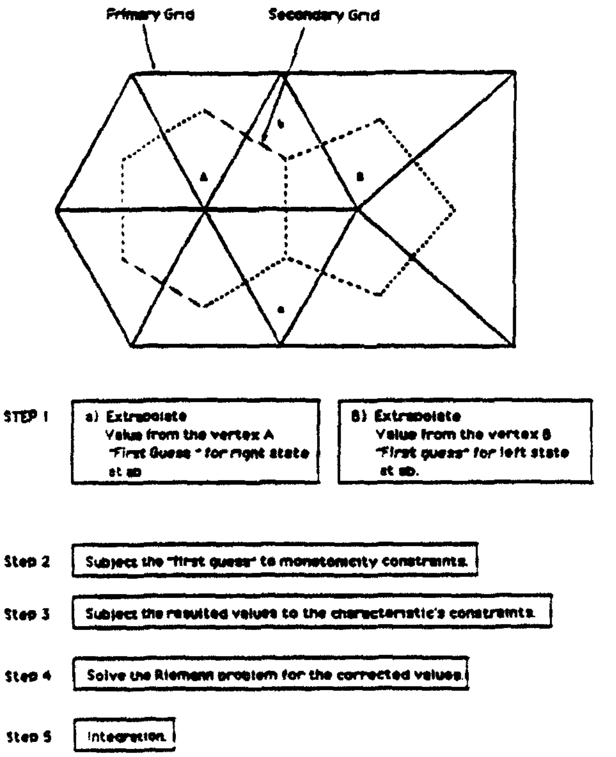


Fig. 4 Grid schematic and outline of steps for second-order Godunov method.

The simulation begins at $t = 0$ when we assume an ideal detonation process has taken place in a stochiometric propane/air mixture. Initially the detonation wave has traveled from the open aft end of the chamber toward the interior with a maximum velocity of 1800 m/s and maximum pressure of 20×10^5 Pa. The distribution of pressure, velocity, and density of the detonation wave is defined through the self-similar solution for a planar detonation wave. These distributions are shown schematically in Fig. 5. The wave was directed toward the interior of the chamber to capture the kinetic energy of the wave and to prolong exposure of the inner chamber walls to the high pressure. In Fig. 6, simulation results are shown at time $t = 0.19$ ms in the form of pressure contours and particle paths from different locations inside and outside the detonation chamber. From the pressure contour plots, we observe that the shock reflection from the inner wall has taken place and detonation products are expanding into the ambient airstream. The flow inside the chamber is choked due to the converging nozzle and the maximum pressure behind the shock is ≈ 8 atm. The pressure inside the chamber is less than 3 atm. The strong expansion of the detonation products into the ambient airstream produces a shock wave with a spherical-like front rapidly decaying in strength. As a result of the interaction of the expanding detonation products with the external flow, a large toroidal vortex is created. The vortex is carried away quickly from the chamber by the external flow and by its own flow momentum.

In Fig. 6a, we also show trajectories of the particles introduced inside the chamber and just above the nozzle. Examination of these trajectories allows us to follow the dynamics of the chamber evacuation and refill. In order to track the detonation products, we initially place marker particles inside the chamber at three cross sections in clusters of four distributed normally to the detonation chamber axis. Each particle has a *different* color; however, particles in the same cluster have the same shade of color. At the three chosen cross sections, we have designated shades of red, yellow, and green for the particles located correspondingly at the left end, center, and beginning of the nozzle cross sections of the chamber. The movement of these particles is shown by connecting them with a continuous line beginning with particle location at $t = 0$ to the present time. In Fig. 6a, we observe that at time $t = 0.19$ ms all particles originally in the nozzle cross section and three of the particles originally in the midsection have left the detonation chamber. However, particles originally introduced on the inner wall of the chamber have only advanced to the nozzle region.

We use a different technique for observing the motion of the ambient gas outside the chamber. Here a cluster of seven

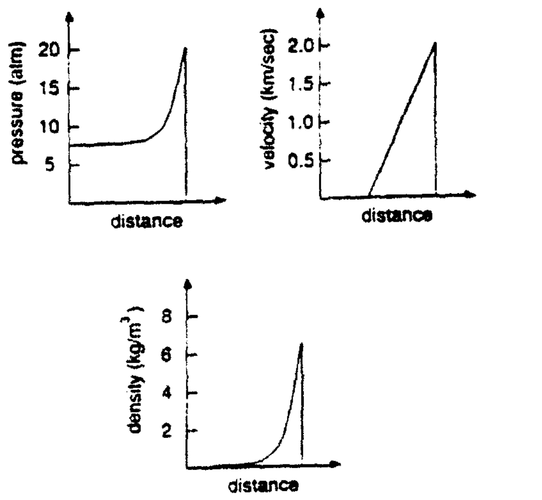
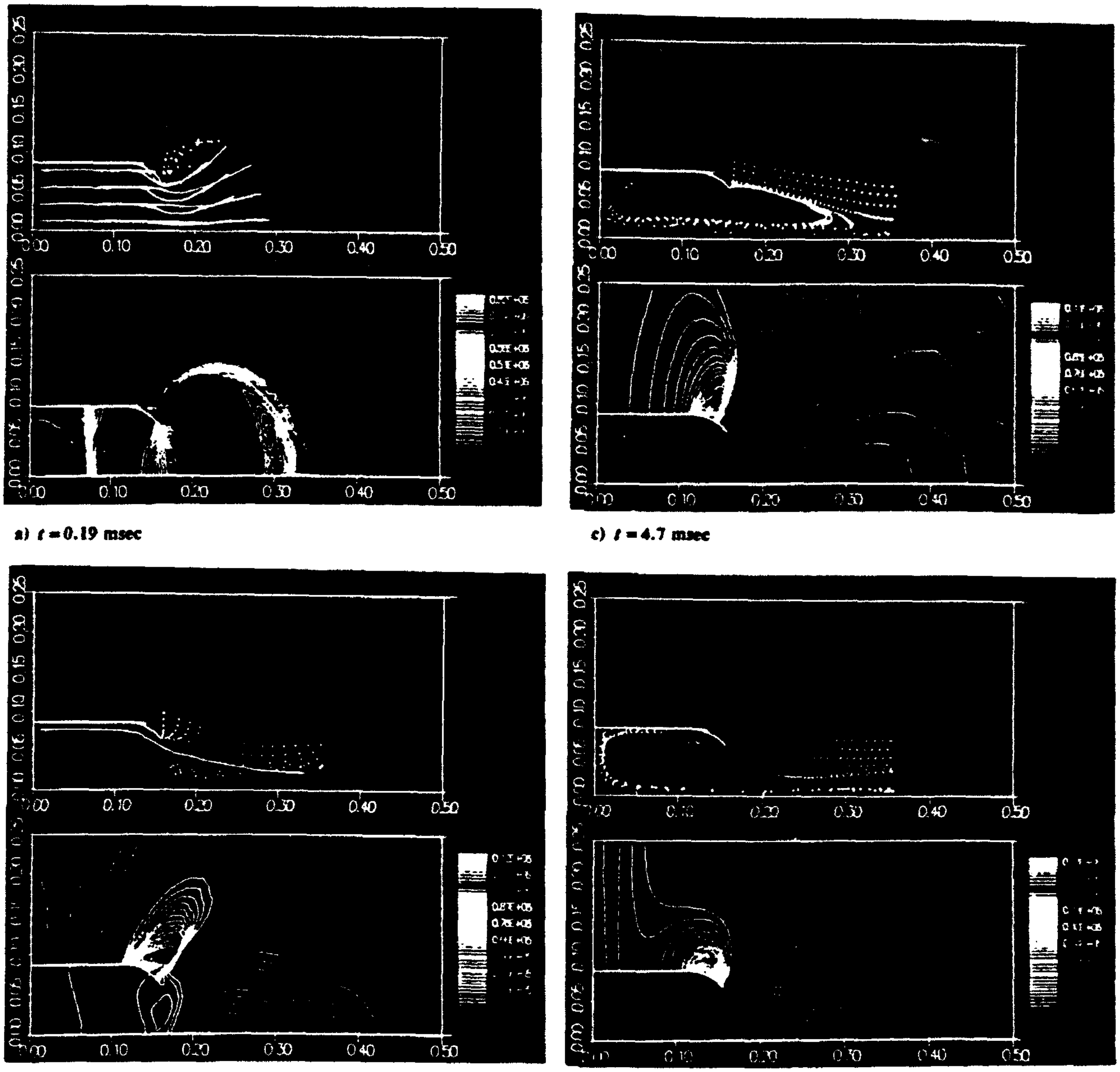


Fig. 5 Distribution of gasdynamic parameters behind the detonation wave according to a one-dimensional self-similar solution.



a) $t = 0.19$ msec

b) $t = 1.7$ msec

c) $t = 4.7$ msec

d) $t = 7.4$ msec, end of first detonation cycle

Fig. 6 Pressure contours and particle paths for various times during the PDEC simulation.

particles is introduced every 0.05 μs in the external flow above the nozzle. All such particles are traced as they move with the flow until they leave the computational domain. At any given time only the current location of the particle is displayed, and since the particles are introduced periodically with time there are a large number of particles to trace. We assign a color to every cluster of external particles to keep track of the time when they were introduced in the calculation. The colors vary from magenta, for those particles introduced early in calculation, to blue, for those introduced shortly before the end of a detonation cycle. In Fig. 6a, corresponding to very early times, only one cluster of external particles is visible. This cluster was introduced at $t = 0$ and is tracking the expanding flow of the detonation products.

In Fig. 6b, the simulation results are shown for $t = 1.7$ ms. The pressure contours show that a shock wave develops at the external edge of the nozzle as a result of a strong expansion of the Mach 0.9 external flow. As a result of overexpansion of the detonation products, the pressure inside the detonation chamber is lower than the ambient pressure, causing the shock to be located lower on the external surface of the nozzle. The external flow about the chamber has a stagnation point on the axis of symmetry downstream at ≈ 25 cm. At this time, it is evident from the particles' trajectories that most of the detonation products have left the chamber. Figure 6b shows one continuous trace of the particles originating at the back wall of the detonation chamber having advanced well ahead of the stagnation point in the external flow.

The marker particles released outside and just above the nozzle's exit show two distinct flow paths. One path takes the flow past the stagnation point to the right of the detonation chamber; this flow path is marked by the four upper particle traces. Another flow path is marked by three lower particle paths released close to the nozzle surface and is deflected toward the detonation chamber exit. Figure 5b shows particles marking this deflected stream approaching the detonation chamber nozzle. The magenta color of these particles indicates they were released at ≈ 0.5 ms.

Figure 6c corresponds to the simulation time $t = 4.7$ ms. The pressure inside the chamber has risen ≈ 1 atm. Higher

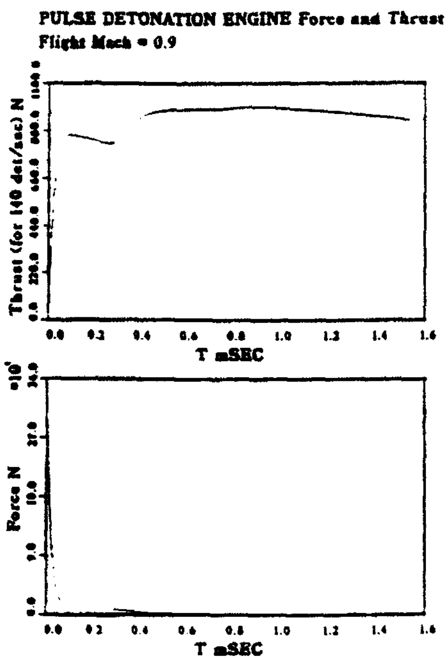


Fig. 7 Thrust and force generated by PDEC as a function of time.

pressure at the chamber exit has resulted in the shock standing on the external surface of the nozzle to move upward. The particles marking the movement of fresh air into the chamber show these to be well inside with some reflecting from the end wall giving a second stagnation point for the reversed fresh airflow.

Figure 6d corresponds to the end of the first cycle when the detonation chamber is filled with fresh charge and ready for the next detonation. In this figure, the particles' paths indicate that the chamber refills in a pattern suitable for fast mixing of the fuel-air mixture. We conjecture then that fuel injection along the chamber axis will promote fast fuel-air mixing. We can see in Fig. 6d that further injection of external air inside the chamber stopped, and from that point on the mixture composition in the chamber will be fixed.

In Fig. 7, the total force and time-averaged thrust generated by the device in the simulations discussed previously are shown as a function of time. The time-averaged thrust is based on the total time for one cycle. As seen in Fig. 7, initially a very large force of $\approx 1.5 \times 10^5$ kg is felt on the end wall of the detonation chamber. This is a result of the inwardly moving detonation wave used in our simulation. Very early during the sequence, this wave reflects from the left wall of the detonation chamber briefly generating a large force. This force rapidly decays and at $t \approx 0.1$ ms changes sign due to interaction of the strong shock wave with the converging nozzle. This effect is noticeable in the thrust data; the average thrust decreases somewhat after reaching levels of ≈ 1980 N. The shock partially reflects from the converging nozzle walls and generates a wave moving to the left wall. The reflected wave thereafter generates positive thrust from $t \approx 0.3$ ms. Finally, thrust levels reach the maximum of ≈ 2200 N and then decay slowly as a result of the cross-sectional drag force. The simulations predict that to sustain this level of thrust will require a detonation frequency of about 150 Hz. All simulations were performed on a Stellar workstation.

Conclusions

The main intent of the present study was to carry out a review of the relevant literature in the area of detonation propulsion, to assess the state of the art, and to recommend future research based on our findings. We have reviewed the literature and presented our summary in the first section of this paper. Our initial conclusion from the review is that there is a substantial body of evidence leading toward the possibility of producing propulsion engines with significant thrust levels based on an intermittent detonation.

Most of the historical attempts at producing thrust based on the intermittent detonation cycle were carried out with the same basic experimental setup; namely, a long straight detonation tube employing forced fuel injection at the closed tube end. We have discussed the many reasons why such a device cannot take proper advantage of the physical processes associated with detonation.

The experiments performed at the Naval Postgraduate School using a self-aspirating mode of operation for a pulsed detonation thruster produced very useful results which, upon further examination, provide us with a route toward practical propulsion engines of variable thrust levels that are both controllable and scalable.

We have explored some of the implications of the possible applications of the self-aspirating detonation engine concept and have developed a suitable numerical simulation code to be used as a design, analysis, and evaluation tool. In fact, the preliminary analysis of a candidate detonation chamber flow was shown to be dominated completely by unsteady gasdynamics. An attempt to understand the flow properties based on any steady-state model or one-dimensional unsteady analytical model will miss such important aspects as fuel-air mixing and shock reflection from internal geometrical obstacle such as the converging nozzle. The unsteady simulation code developed during the course of our study is a necessary tool that we plan to use in a study leading to a feasible prototype engine design realizing the full potential of the intermittent detonation process.

Acknowledgments

The work reported here was supported by the Defense Advanced Research Projects Agency under Contract N66001-88-D-0088. The authors express appreciation to Adam Drobot and Aharon Friedman for helpful suggestions and advice.

References

[1]Stodola, A., *Steam and Gas Turbines*, McGraw-Hill, New York, 1927.

[2]Zipkin, M. A., and Lewis, G. W., "Analytical and Experimental Performance of an Explosion-Cycle Combustion Chamber of a Jet Propulsion Engine," NACA TN-1702, Sept. 1948.

[3]Shultz-Grunow, F., "Gas-Dynamic Investigation of the Pulse-Jet Tube," NACA TM-1131, Feb. 1947.

[4]Zinn, B. T., Miller, N., Carvelho, J. A., and Daniel, B. R., "Pulsating Combustion of Coal in Rijke Type Combustor," *19th International Symposium on Combustion*, Combustion Inst., Pittsburgh, PA, 1982, pp. 1197–1203.

[5]Hoffmann, N., "Reaction Propulsion by Intermittent Detonative Combustion," Ministry of Supply, Volkenrode Translation, 1940.

[6]Nicholls, J. A., Wilkinson, H. R., and Morrison, R. B., "Intermittent Detonation as a Thrust-Producing Mechanism," *Jet Propulsion*, Vol. 27, 1957, pp. 534–541.

[7]Dunlap, R., Brehm, R. L., and Nicholls, J. A., "A Preliminary Study of the Application of Steady State Detonative Combustion of a Reaction Engine," *Jet Propulsion*, Vol. 28, 1958, pp. 451–456.

[8]Nicholls, J. A., Gullen, R. E., and Ragland, K. W., "Feasibility Studies of a Rotating Detonation Wave Rocket Motor," *Journal of Spacecraft and Rockets*, Vol. 3, 1966, pp. 893–898.

[9]Adamson, T. C., and Olsson, G. R., "Performance Analysis of a Rotating Detonation Wave Rocket Engine," *Astronautica Acta*, Vol. 13, 1967, pp. 405–415.

[10]Shen, P. I., and Adamson, T. C., "Theoretical Analysis of a Rotating Two-Phase Detonation in Liquid Rocket Motors," *Astronautica Acta*, Vol. 17, 1972, pp. 715–728.

[11]Krzycki, L. J., "Performance Characteristics of an Intermittent Detonation Device," U. S. Naval Ordnance Test Station, China Lake, CA, Navweps Rept. 7655, 1962.

[12]Matsui, H., and Lee, J. H., "On the Measure of the Relative Detonation Hazards of Gaseous Fuel-Oxygen and Air Mixtures," *Seventeenth Symposium (International) on Combustion*, Combustion Inst., Pittsburgh, PA, 1978, pp. 1269–1280

[13]Korovin, L. N., Losev, A., Ruban, S. G., and Smekhov, G. D.,

"Combustion of Natural Gas in a Commercial Detonation Reactor," Fizika Gor. Vzryva, Vol. 17, No. 3, 1981, p. 86.

[14]Smirnov, N. N., and Boichenko, A. P., "Transition from Deflagration to Detonation in Gasoline-Air Mixtures," Fizika Gor. Vzryva, Vol. 22, No. 2, 1986, pp. 65–67.

[15]Lobanov, D. P., Fonbershtein, E. G., and Ekomasov, S. P., "Detonation of Gasoline-Air Mixtures in Small Diameter Tubes," Fizika Gor. Vzryva, Vol. 12, No. 3, 1976, p. 446.

[16]Back, L. H., "Application of Blast Wave Theory to Explosive Propulsion," *Acta Astronautica*, Vol. 2, No. 5/6, 1975, pp. 391–407.

[17]Varsi, G., Back, L. H., and Kim, K., "Blast Wave in a Nozzle for Propulsion Applications," *Acta Astronautica*, Vol. 3, 1976, pp. 141–156.

[18]Kim, K., Varsi, G., and Back, L. H., "Blast Wave Analysis for Detonation Propulsion," *AIAA Journal*, Vol. 10, No. 10, 1977, pp. 1500–1502.

[19]Back, L. H., Dowler, W. L., and Varsi, G., "Detonation Propulsion Experiments and Theory," *AIAA Journal*, Vol. 21, No. 10, 1983, pp. 1418–1427.

[20]Eidelman, S., and Shreeve, R. P., "Numerical Modeling of the Nonsteady Thrust Produced by Intermittent Pressure Rise in a Diverging Channel," American Society of Mechanical Engineers, *Multidimensional Fluid Transient*, FED-Vol. 18, 1984, p. 77.

[21]Eidelman, S., "Rotary Detonation Engine," U.S. Patent 4 741 154, 1988.

[22]Helman, D., Shreeve, R. P., and Eidelman, S., "Detonation Pulse Engine," AIAA Paper 86-1683, June 1986.

[23]Monks, S. A., "Preliminary Assessment of a Rotary Detonation Engine Concept," M.Sc. Thesis, Naval Postgraduate School, Monterey, CA, Sept. 1983.

[24]Camblier, T. L., and Adelman, N. G., "Preliminary Numerical Simulations of a Pulsed Detonation Wave Engine," AIAA Paper 88-2960, Aug. 1988.

[25]Lohner, R., Morgan, K., and Zienkiewicz, D. C., "Finite-Element Methods for High Speed Flows," AIAA Paper 85-1531, July 1985.

[26]Lohner, R., and Morgan, K., "Improved Adaptive Refinement Strategies for Finite-Element Aerodynamic Computations," AIAA Paper 86-0499, Jan. 1986.

[27]Barth, T. J., and Jespersen, D. C., "The Design and Application o Jpwind Schemes on Unstructured Meshes," AIAA Paper 89-0366, Jan 1989.

[28]Eidelman, S., Collela, P., and Shreeve, R. P., "Application of the Godunov Method and Its Second-Order Extension to Cascade Flow Modelling," *AIAA Journal*, Vol. 22, No. 1, 1984, p. 10.

[29]Baker, T. J., "Developments and Trends in Three-Dimensional Mesh Generations," Transonic Symposium, NASA Langley Research Center, Hampton, VA, 1988.

[30]Jameson, A., Baker, T. J., and Weatherill, N. P., "Calculation of Inviscid Transonic Flow Over a Complete Aircraft," AIAA Paper 86-0103, Jan. 1986.

[31]Greengard, L., and Rokhlin, V., "A Fast Algorithm for Particle Simulations," *Journal of Computational Physics*, Vol. 73, 1987, pp. 325–348.

[32]Eidelman, S., Collela, P., and Shreeve, R. P., "Application of the Godunov Method and Its Second Order Extension to Cascade Flow Modeling," *AIAA Journal*, Vol. 22, No. 1, 1984, p. 10.

[33]van Leer, B., "Towards the Ultimate Conservative Difference Scheme, V. A Second Order Sequel to Godunov's Method," *Journal of Computational Physics*, Vol. 32, 1979, pp. 101–136.

[34]Collela, P., and Woodward, P., "The Piecewise Parabolic Method (PPM) for Gasdynamical Simulations," *Journal of Computational Physics*, Vol. 54, 1984, pp. 174–201.

[35]Barth, T. J., and Jespersen, D. C., "The Design and Application of Upwind Schemes on Unstructured Meshes," AIAA Paper 89-0366, Jan. 1989.

[36]Glaz, H. M., Collela, P., Glass, I. I., and Deschambault, R. L., "A Detailed Numerical, Graphical, and Experimental Study of Oblique Shock Wave Reflections," Defense Nuclear Agency, DNA-TR-86-365, 1986.

Shock Waves (1991) 1: 197–203

Shock Waves

Numerical and analytical study of transverse supersonic flow over a flat cone

D.L. Book[1], S. Eidelman[2], I. Lottati[2] and X. Yang[2]

[1] Code 4405, Naval Research Laboratory, Washington, DC 20375
[2] Science Applications International, McLean, VA 22102

Received February 8, 1991; accepted March 21, 1991

Abstract. Quasisteady supersonic flow over a flat cone on a plane surface is studied. A formula is derived for the angle through which the flow lines turn at the cone. The results are used to justify the use of two-dimensional simulations of the flow. Peak pressures and total impulses are obtained numerically for various cone angles.

Key words: Cone, Euler equation, Mach reflection, CFD, Supersonic flow

1. Introduction

The purpose of this study is to determine the maximum pressure on the surface of a flat cone (one for which the height is much less than the diameter) in the quasi-uniform flow behind a strong blast wave propagating at right angles to the axis of the cone. If the cone is small compared with the radius of the blast wave, the undisturbed flow is approximately rectilinear. First, the blast wave passes over the cone and an unsteady load builds up on the surface. In general the shock will undergo Mach reflection over at least part of the surface of the cone, the extent depending on the cone angle α, the adiabatic index γ, and the Mach number M. After a short transitional stage the cone will then be subject to the quasisteady supersonic flow field behind the blast front. (For a strong blast wave in air, the pressure drops to one-half the peak value about one-tenth of the way back from the front toward the origin of the blast (Sedov 1959). Thus, if the blast center is located ~ 100 radii from the cone, the pressure arriving at the cone is reduced to half its initial value ~ 10 radii behind the front.) The post-shock flow velocity varies on the same scale. We would like to find whether the pressure on the cone reaches its maximum during the quasisteady or the unsteady regime of the flow and determine its magnitude.

The cone, shown schematically in Fig. 1, is located on a plane surface. We take its axis to be normal to the surface, and we model the front of the spherical blast wave as a planar shock wave propagating normally to this axis. This is a reasonable approximation when the distance to the blast center is much larger than the radius, i.e., in the same limit for which we can assume that the state behind the front is uniform. The flow over the cone is substantially three-dimensional. The only symmetry is with respect to inversion about the midplane (the plane through the cone axis and parallel to the flow). In the general case in which the cone axis is not normal to the plane, the problem is totally asymmetrical.

Previous studies of the effect of supersonic flows on conical bodies have focused primarily on situations in which the flow is parallel or nearly parallel to the axis of the cone. Those results are applicable to, e.g., the aerodynamical effects associated with the nose cones of re-entry vehicles. In contrast, the problem we are considering may be regarded as an idealized model of the interaction between a blast wave and a ground or shipboard structure. It can also model the flow over a bump or a housing on the skin of a supersonic aircraft or missile. The results may thus be relevant to both damage studies and flight characteristics.

A number of experimental studies related to the problem of oblique supersonic flow over a cone have been carried out, beginning at least three decades ago (Tracy 1963; Damkevala and Zumwalt 1968). Most of this work has dealt with small deviations from axisymmetry, although angles of attack as large as 30° have been studied (Yahalom 1971). Less experimental effort seems to have been devoted to transverse flows (angle of attack equal to 90°). Likewise, theoretical studies by Goman and Davydov (1975) and Gusarov et al. (1979) have concentrated on small deviations from conical shapes in axisymmetric flow. Numerical simulations have been car-

Offprint requests to: D.L. Book

ried out at large angles of attack (30°–50°) by Fletcher and Holt (1976). These results are useful, and the same techniques can be applied to transverse flows, but they have definite limitations. At the Mach numbers investigated (M $\lesssim$ 6–8) the flow is strongly conditioned by the presence of a viscous boundary layer. This necessitates solution of the Navier-Stokes equations instead of the Euler equations, and it may be necessary to incorporate a turbulence model as well. In three dimensions it is difficult to obtain good resolution even for inviscid flow; the presence of a thin boundary layer makes the problem even more formidable.

In the next section we determine the streamlines associated with transverse flow over a cone in the Newtonian approximation, i.e., assuming that the streamlines follow the contours of the body surface. We show that for a flat cone the streamlines deviate very little from the vertical plane in which they were propagating before reaching the cone. We use this result to argue that the flow over the cone can be accurately modeled by treating each cross section made by a vertical plane separately, i.e., by solving a series of two-dimensional problems. In the section following that, we describe the results of such calculations. For this purpose we use an Euler code, which is only valid at low flow Mach numbers (M$\lesssim$5). In our calculations the shock Mach number equals 25, but the Mach number of the flow in the heated region behind the shock is ~ 3, so we are justified in ignoring viscous effects. At higher values of M our results are at least indicative and can be expected to yield accurate values of the peak pressures on the cone. (Of course the reduction of the problem to two dimensions is a consequence of the cone geometry and would be equally useful for Navier-Stokes applications.) We show that these results can be combined to draw a picture of most of the flow field. In the final section we summarize our conclusions.

2. Streamline trajectories

If the flow deflected by a solid object remains supersonic after deflection, the angle between the flow direction and the surface determines the flow parameters behind the shock for given inlet flow parameters. For shock Mach numbers M$\gtrsim$10, the shock angle is small and the deflected flow on the upwind side closely follows the form of the deflecting object. The streamlines are determined by the condition that the angle through which they are deflected be as small as possible. We would like to analyze how this deflection angle varies on the surface of the cone shown in Fig. 1. Based on this analysis we can estimate most of the characteristics of steady supersonic flow directed transversely toward the cone.

The equation for the frustrum of a cone with the geometrical center of its base located at the center of coordinates (see Fig. 1) is

$$(x^2+y^2)\tan^2\alpha = (z-h)^2, \tag{1}$$

where h is the height of the cone and α is the angle between the side of the cone and the base. The angle

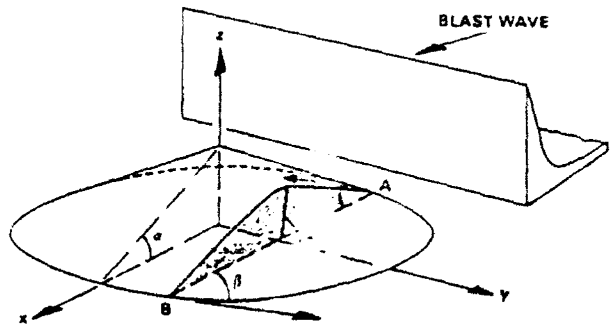


Fig. 1. Schematic of the model. The z axis coincides with the axis of the cone and the flow is taken in the direction of the positive x axis. The angles α, β, γ, and ω are defined in the text

δ between the propagation direction $\mathbf{n}$ (taken to be the positive x-direction) and the deflected streamline at the leading edge of the cone, which determines the shock strength, is bounded above by the angles between $\mathbf{n}$ and the conic sections in the x–y and x–z planes.

The cross sections of the cone parallel to the x–y plane are circles given by

$$x^2+y^2 = (z_0-h)^2/\tan^2\alpha \equiv r^2, \tag{2}$$

where z_0 is constant for each particular cross section and r is the radius of the circle. The angle β between $\mathbf{n}$ and the tangent to this circle at the point with ordinate y is given by

$$\tan\beta = \frac{\partial y}{\partial x} = -\frac{x}{(r^2-x^2)^{1/2}} = -\frac{(r^2-y^2)^{1/2}}{y}. \tag{3}$$

The sign is chosen so that positive values of β correspond to negative values of x (i.e., the upwind side).

The cross sections parallel to the x–z plane form hyperbolas on the surface of the cone. The equation for this family of curves is

$$\frac{(z-h)^2}{\tan^2\alpha} - x^2 = {y_0}^2, \tag{4}$$

where y_0 is constant for each particular cross section. The angle γ between $\mathbf{n}$ and the tangent to this hyperbola is given by

$$\begin{aligned}\tan\gamma = \frac{\partial z}{\partial x} &= \tan\alpha/\left(1+{y_0}^2/x^2\right)^{1/2} \\ &= \left(1-{y_0}^2/r^2\right)^{1/2}\tan\alpha\end{aligned} \tag{5}$$

Let us examine now how $\tan\beta$ and $\tan\gamma$ vary on the intersection of the cone with the x–y plane when y changes from 0 to $\pm R$, where $r = R = h/\tan\alpha$. From (3), $\tan\beta$ approaches ∞ and 0, i.e., $\beta = 90°$ and $\beta = 0°$, in the limits $y \to 0$ and $y \to \pm R$, respectively. From (5), $\tan\gamma$ approaches $\tan\alpha$ and 0 in the same limits, corresponding to $\gamma = \alpha$ and $\gamma = 0°$.

Comparing (3) and (5), we readily conclude that for $\tan\alpha < 1$

$$\begin{aligned} &\tan\gamma < \tan\beta, \quad 0 < y < R; \\ &\tan\gamma = \tan\beta = 0, \quad y = R, \end{aligned} \tag{6}$$

and for $\tan\alpha > 1$

$$\begin{aligned} &\tan\gamma < \tan\beta, \quad 0 < y < R/\tan\alpha; \\ &\tan\gamma = \tan\beta, \quad y = R/\tan\alpha; \\ &\tan\gamma > \tan\beta, \quad R/\tan\alpha < y < R; \\ &\tan\gamma = \tan\beta = 0, \quad y = R. \end{aligned} \tag{7}$$

Thus, at any point with $x \le 0$ on the cone specified by (1) for $\tan\alpha < 1$, the propagation vector $\mathbf{n}$ makes a smaller angle with the cone in the cross section parallel to the x–z plane than in the one parallel to the x–y plane. For supersonic flow over the cone shown in Fig. 1, condition (6) implies that the velocity vectors behind the shock front in the region of compression of the flow will always be directed over the cone and not around it.

Now we consider intermediate cross sections of the cone, obtainable by rotating through an angle ω about the line AB defined by the intersection of the x–y plane and a plane parallel to the x–z plane. We would like to find the minimum angle between $\mathbf{n}$ and the tangent in these cross sections when ω varies from 0° (cross section parallel to the x–y plane) to 90° (cross section parallel to the x–z plane).

This family of cross sections is defined by (1) together with the equation of the cross-section plane,

$$z - Z = (y - Y)\tan\omega. \tag{8}$$

where ω is the angle between the cross-section plane. We restrict ourselves to points lying in the x–y plane as shown in Fig. 1, since the bow shock produced by the interaction between the flow and the cone will either be attached at this point or will stand off slightly ahead of the cone. The coordinates of the point where the flow encounters the cone are $X = (R^2 - Y^2)^{1/2}, Y$, and $Z = 0$. The tangent line is the intersection of this plane and the plane tangent to the cone at (X, Y, Z). The equation of the latter is obtained from (1):

$$(xX + yY)\tan^2\alpha + h(z - h) = 0. \tag{9}$$

Solving (8) and (9) simultaneously yields the equations describing the tangent line:

$$z = (y - Y)\tan\omega = \frac{-(x - X)X\tan^2\alpha\tan\omega}{h\tan\omega + Y\tan^2\alpha}. \tag{10}$$

The angle δ between this line and $\mathbf{n}$ is given by

$$\begin{aligned} \tan\delta &= \frac{\left[(y - Y)^2 + z^2\right]^{1/2}}{x - X} \\ &= \frac{\left(h^2 - Y^2\tan^2\alpha\right)^{1/2}\tan\alpha}{h\sin\omega + Y\tan^2\alpha\cos\omega} \equiv f(\omega). \end{aligned} \tag{11}$$

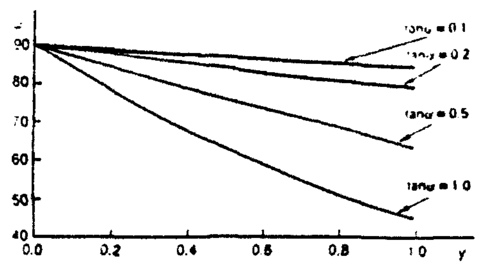


Fig. 2. Value of angle ω which minimizes deflection angle as a function of Y/R

Now we look for the extrema of $f(\omega)$ when ω varies from 0° to 90°:

$$\frac{df}{d\omega} = \frac{\tan\delta\,(Y\tan^2\alpha\sin\omega - h\cos\omega)}{h\sin\omega + Y\tan^2\alpha\cos\omega}, \tag{12}$$

which vanishes only for

$$\tan\omega = \frac{h}{Y\tan^2\alpha} \equiv \tan\omega_{\min}. \tag{13}$$

It is easy to show that (13) defines the minimum of f. Substituting (13) into (11), we find

$$\tan\delta_{\min} = \frac{\left(h^2 - Y^2\tan^2\alpha\right)^{1/2}\tan\alpha}{\left(h^2 + Y^2\tan^4\alpha\right)^{1/2}}, \tag{14}$$

which determines the angle $\delta_{\min}$ through which the streamline of the supersonic flow behind the shock wave turns as a function of Y.

From (13) we see that $\tan\omega > 1$ holds for $\tan\alpha < 1$, since $Y \le h/\tan\alpha$. Figure 2 shows how $\omega_{\min}$ changes when Y varies from 0 to R for various values of $\tan\alpha$. This figure implies that for flat cones the supersonic flow will be almost parallel to the x–z plane.

Another way to reach the same conclusion is by finding the maximum y-displacement of a streamline from its original trajectory. This occurs for $x = 0$, when the trajectory reaches its highest point on the surface of the cone. The tangent to the cone at $x = 0$, i.e., the section of the cone in the y–z plane, is described by

$$y\tan\alpha + z = h. \tag{15}$$

Solving this equation together with

$$z = (y - Y)\tan\omega_{\min} = \frac{h(y - Y)}{Y\tan^2\alpha}, \tag{16}$$

we obtain

$$y = \frac{hY\sec^2\alpha}{h + Y\tan^3\alpha} \equiv y_0. \tag{17}$$

Maximizing $\Delta y = y_0 - Y$ with respect to Y, we find that the largest value of Δy occurs for

$$Y = \frac{h\cos^2\alpha}{\sin\alpha(1 + \cos\alpha)} = \frac{R\cos\alpha}{1 + \cos\alpha} \tag{18}$$

and equals

$$(\Delta y)_{\max} = \frac{h \sin\alpha \cos\alpha}{(1+\cos\alpha)^2} = \frac{h \tan^2 \alpha/2}{\tan\alpha}. \tag{19}$$

For $\alpha \ll 1$ we have $Y_{\max} \approx R/2$ and $(\Delta y)_{\max} \approx h\alpha/4$.

In summary, the vertical deflection of the streamlines over a cone of small edge angle α is of order α, while the horizontal deflection is of order α^2. For $\alpha \lesssim 0.1$ we see that the flow over the cone deviates from the vertical plane in which it starts out by an amount 0.01. Thus the compression region of supersonic flow over flat cones can be calculated accurately by modeling the flow over separate cross sections of the cone made by planes parallel to the x–z plane. A wedge, being two-dimensional, is easier to model than a cone. For this reason we carried out several calculations of the interaction between blast waves and wedges, described in the next section.

We can also conclude that the maximum change in the direction of a streamline for such cones will be α. If the shock undergoes regular reflection, uniform supersonic flow over a wedge with base angle α gives an upper bound for the pressure on the cone. Where Mach reflection occurs it is necessary to model the transient regime, as the pressure peaks associated with the Mach stem and the contact surface could conceivably be larger.

3. Numerical modeling

Let us consider a cone with $\alpha = 10°$ ($\tan\alpha = 0.176$) at the base. According to the analysis in the preceding section of a transversely directed supersonic flow over a cone, an upper limit can be obtained by modeling the same supersonic flow over a wedge with opening angle α.

Here we present the result obtained by numerically solving the equations for the flow over the wedge when it is loaded by a passing blast wave. For the simulation we used the Fast Unstructured-Grid Second-Order Godunov Solver, described by Eidelman and Lottati (1990). This code, which is based on a second-order Godunov method (Eidelman et al. 1984), provides a method for solving the Euler equations of gasdynamics on unstructured grids with arbitrary connectivity. The use of a data structure with only one level of indirectness leads to an easily vectorized and parallelized code with low memory requirements and high computational efficiency. The algorithm has been tested for performance and accuracy over a wide range of Mach numbers and geometrical situations, and has demonstrated robustness without the need for any adjustable parameters. It can be implemented in either a triangle- or vertex-based form; experience with the method has shown that extremely low levels of artificial viscosity can be achieved using the triangle-based version of the method. Direct dynamic refinement of the grid (Eidelman and Lottati 1990) allows automatic adaptation to the front of the moving blast wave. This refinement guarantees that the associated highly inhomogeneous pressure and density features are accurately tracked.

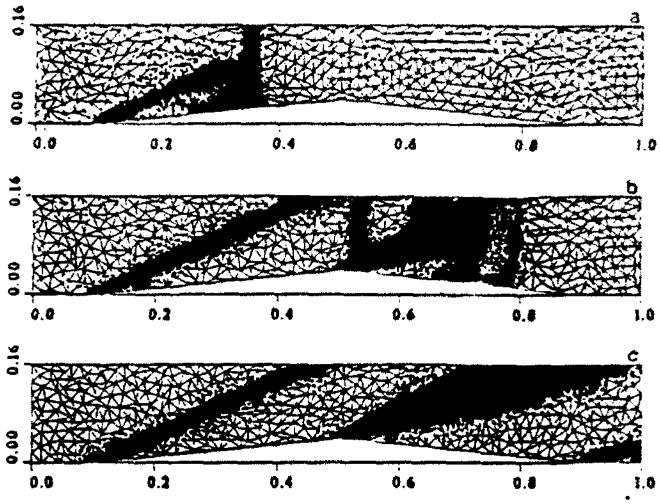


Fig. 3. Unstructured grid for $\alpha = 10°$ at times (a) 35 μs, (b) 55 μs, and (c) 130 μs, associated with the density and pressure contour plots of Figs. 4 and 5. Distances are in meters. These reproductions are unable to resolve the smallest triangles, which show up as dark regions roughly coincident with the locations of gasdynamic discontinuities

For the initial conditions in the computational domain we assume air at standard temperature and pressure. At $t = 0$, a strong (M = 25) blast wave, propagating to the right, is located at the left boundary. We assume that the blast wave is "square" and that conditions at the left boundary of the computational domain remain constant for the whole time of the simulation. A constant value of $\gamma = 1.2$ was used (appropriate to flow behind shocks with this value of M on account of real-gas effects). For these values of α, γ, and M, shock tube measurements described by Glass (1987) of diffraction over a wedge indicate that double Mach reflection should occur.

Figure 3 shows the computational grid at various times t: (a) at $t = 35\,\mu$s, shortly before the blast front reaches the apex of the wedge, located at a horizontal distance $l = 1$ m from the corner; (b) at $t = 55\,\mu$s, just after it passes the apex; and (c) at $t = 130\,\mu$s, after the leading shock has exited from the computational domain and a quasisteady state has developed. The highly refined portions of the grid follow shock fronts, contact discontinuities, etc. The numbers of vertices shown are 4166, 11785, and 10959, respectively, reflecting the complexity of the corresponding states, i.e., the amount of structure in the gasdynamic processes present.

Figure 4 shows contours of density scaled by the ambient density $\rho_0 = 1.29\,\mathrm{kg\,m^{-3}}$ at the same times as in Fig. 3. In the first frame the flow is still identical with that for a shock reflecting from a single wedge with opening angle α, and therefore is evolving self-similarly (Glass 1987). The first Mach stem and incident shock are clearly defined. The associated contact surface is barely discernible, both because the contour levels are bunched near the much larger jumps at the shocks and because at very high Mach numbers the slip line is found quite close

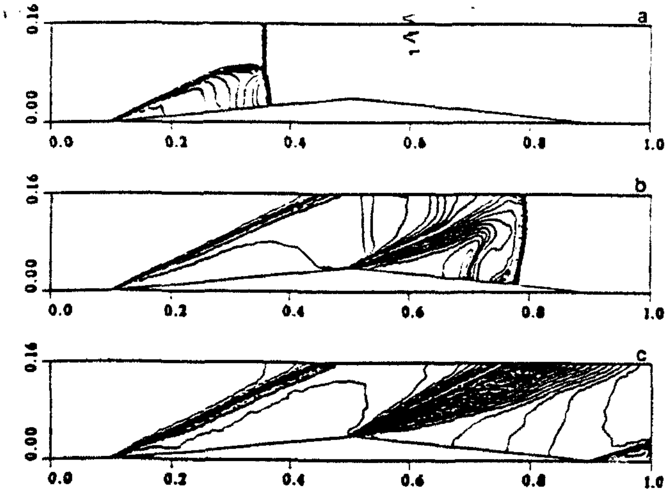


Fig. 4. Scaled density contours for $\alpha = 10°$ at times (a) 35 μs, (b) 55 μs, and (c) 130 μs. Thirty-five contour levels are plotted, with ρ/ρ_0 varying from 1.0 to 13.5. They are concentrated at the large density jumps in the strong shocks. This causes the shocks to be emphasized more than the contact discontinuity, where the density change is relatively small. The structure of the Mach reflection is discernible only in the earliest frame. In the final frame the flow has become essentially steady

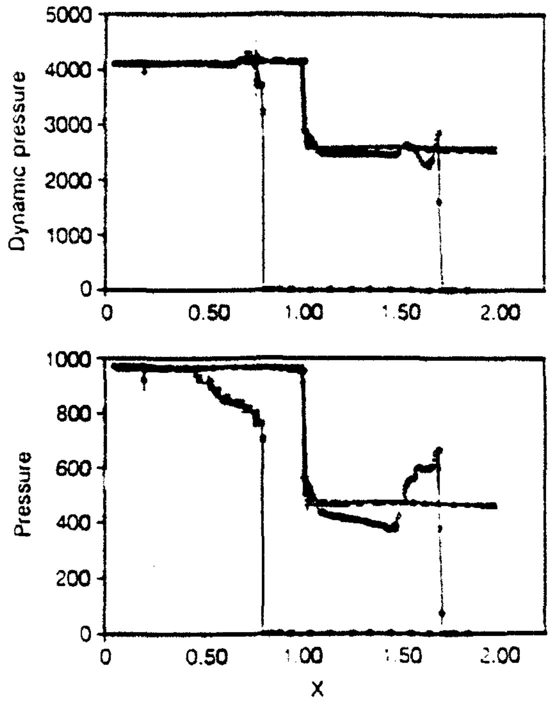


Fig. 5. Scaled dynamic and static pressure on the wedge surface in the case $\alpha = 10°$ as functions of the distance for times 35 μs (□), 55 μs (o), and 130 μs (△). Ahead of the shock these quantities have their ambient values; far behind the shock they become essentially steady

to the Mach stem (Glaz et al. 1985). In Fig. 4b the front has passed the apex and the evolution is no longer self-similar. The flow behind the front expands through an expansion fan attached to the corner. Also clearly visible is the recompression shock two-thirds of the way from the corner to the front. This shock, which serves to reconcile the high pressures in the region following the Mach stem with the lower values appropriate to the expanded flow downstream from the corner, is propagating backward but is being swept to the right by the strong flow behind the leading shock. The triple points have moved far above the cone and no longer appear on the grid. Note that the supersonic outflow boundary condition imposed at the top of the mesh allows material and waves to pass out of the system without reflecting and without causing other signals to propagate back inside. Figure 4c depicts the flow at late times, when transients have essentially disappeared. The only gasdynamic features visible are shocks at the leading and trailing edges and the expansion fan.

Figure 5 shows traces of the static and dynamic pressure scaled by the ambient pressure $p_0 = 101.3$ kPa along the top surface of the wedge as functions of the horizontal distance x in meters at the three specified times. Ahead of the blast these quantities are at ambient levels, they rise sharply when the shock sweeps past, fluctuate, and finally reach their asymptotic values. Note that, as is seen experimentally in shock-tube studies (Glass 1987), the pressure on the surface of the wedge is highest at the leading edge. It is also important to notice that, although these traces exhibit considerable structure (especially the static pressure), the maximum values of the pressure and density for the transient stages are always smaller than those in the quasisteady flow regime. At the same time, values of the Mach number in the transitional stage can be higher than in the quasisteady state. For our case, however, the maximum Mach number is at most 10% higher than the steady-state value. This shows that the maximum force is applied to any point on the surface of the wedge in the quasisteady state.

Figure 6 shows as functions of time the drag and lift coefficients, defined by

$$C_D = \frac{\int p_{\parallel} dx}{\rho_\infty u_\infty^2 l} \tag{20}$$

and

$$C_L = \frac{\int p_{\perp} dx}{\rho_\infty u_\infty^2 l}. \tag{21}$$

Here $p_{\parallel} = p\cos\theta$ and $p_{\perp} = p\sin\theta$ are the horizontal and vertical components of the pressure in terms of the angle θ between the normal to the surface and the x axis, the integrals are carried out over the surface of the wedge, and ρ_∞ and u_∞ are the density behind the undisturbed shock front. The lift grows monotonically, but the drag first rises, then drops to its quasisteady value. The decrease results from the increase in pressure on the trailing side of the wedge when the shocked air reaches that side.

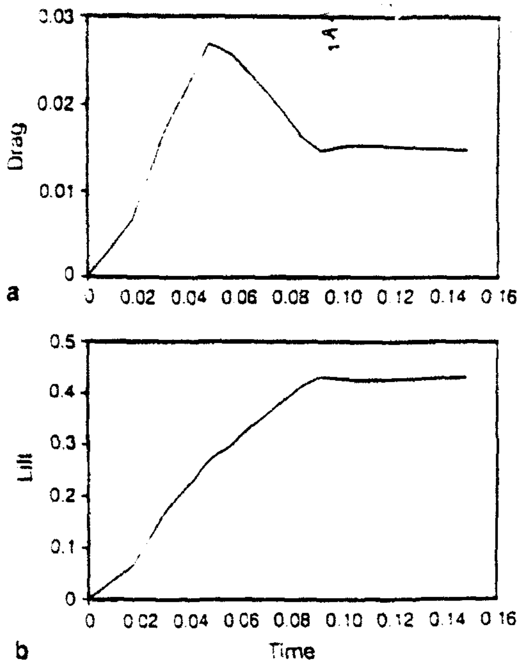


Fig. 6. Lift and drag coefficients for $\alpha = 10°$ as functions of time

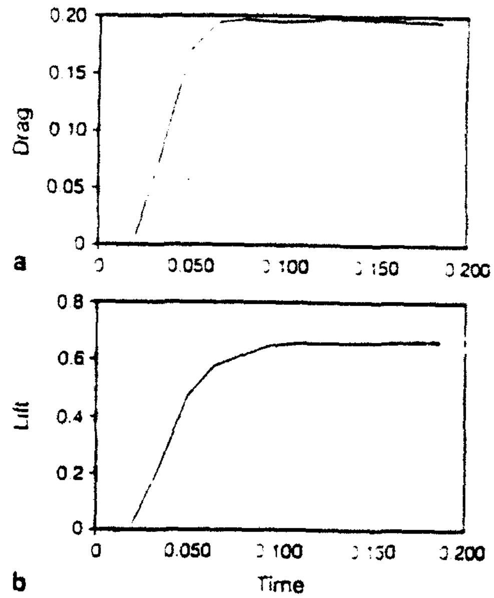


Fig. 8. Lift and drag coefficients for $\alpha = 20°$ as functions of time (in milliseconds)

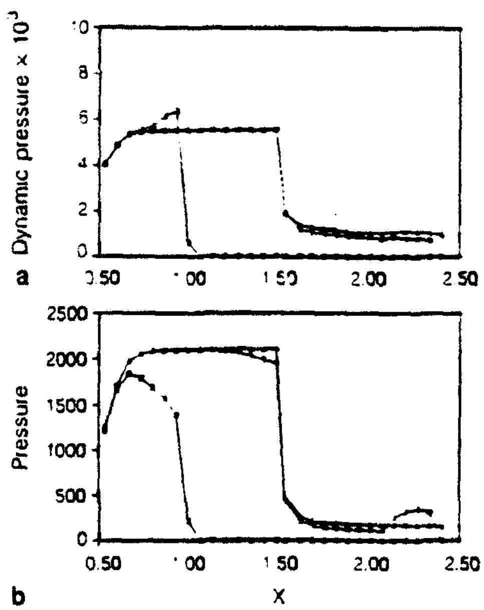


Fig. 7. Scaled dynamic and static pressure on the wedge surface in the case $\alpha = 20°$ as functions of the distance for times 34 μs (□), 94 μs (○), and 168 μs (△)

To learn how sensitive the flow is to the wedge angle, we carried out a second calculation with $\alpha = 20°$ ($\tan\alpha = 0.364$) and the other parameters unchanged. This calculation was done with a coarser grid than the previous one, with triangles about a factor of three larger. Most of the features resembled those of the first case. For example, the traces of the static and dynamic pressure along the top surface, shown in Fig. 7, are qualitatively similar to those for $\alpha = 10°$. One difference is that the drag rises monotonically as a function of time (Fig. 8), rather than decreasing after the shock has passed. This is because the expansion fan attached to the top of the cone is stronger and a low-pressure "bubble" forms on the lee side.

We carried out additional calculations with other values of the parameters. As long as α was small and M was large the results resembled those discussed above. They are not described here, since in no case did the transient pressures exceed those in the quasisteady state, nor were the features in the flow qualitatively different.

4. Conclusions

From the foregoing treatment it is clear that the same modeling technique can be used to determine the pressure distribution in cross sections other than the midplane. So long as $|y|$ does not approach R, the deflection is mostly vertical. The corresponding profile is now a hyperbola, but it differs noticeably from a wedge only near the top. The principal difference is in the expansion wave at the top, which becomes broader than the centered rarefaction wave seen above. For larger values of y the cross section of minimum deflection, found by solving (1) and (8) together, is more rounded at the top but the leading edge of this hyperbolic "wedge" has a smaller angle. By combining pressure distributions at several representative values of y we can find the pressure loading over the entire cone. The picture breaks down only at the lateral extremities of the cone ($|y| \sim R$).

On the basis of qualitative arguments and numerical simulation, our study of the flow resulting from a blast wave propagating transversely over a cone leads to the following conclusions:

1. Flow over a cone with small base angle can be accurately simulated by individually modeling the two-dimensional flows over cross sections of the cone made by vertical planes perpendicular to the shock front.
2. The maximum load on the cone can be calculated from the solution of the flow over the cross section determined by the plane through the cone axis (Fig. 1).
3. In this solution the pressure attains its maximum as a function of time in the quasisteady supersonic regime established after the front has passed.

Acknowledgements. This work was sponsored in part by the Defense Nuclear Agency. The work of the first author was supported by the Naval Research Laboratory.

References

Damkevala RJ, Zumwalt GW (1968) Technique for studying interactions between a body moving at supersonic speeds and blast waves approaching obliquely. Rev Sci Instr 39:1254-1256

Eidelman S, Colella P, Shreeve RP (1984) Application of the Godunov method and its second-order extension to cascade flow modeling. AIAA J 22:1604-1615

Eidelman S, Lottati I (1990) Reflection of the triple point of Mach reflection in planar and axisymmetric converging channels. In: Reichenbach H, (Ed) Proc 9th Mach Reflection Symp Freiburg FGR

Fletcher CJ, Holt M, (1976) Supersonic viscous flow over cones at large angles of attack. J Fluid Mech 74:561-591

Glass II (1987) Some aspects of shock-wave research. AIAA J 25:214-228

Glaz HM, Colella P, Glass II, Deschambault RL (1985) A numerical study of oblique shock-wave reflections with experimental comparisons. Proc R Soc London A398:117-140

Goman OG, Davydov VI (1975) Determination of aerodynamic charateristics of cone with arbitrary small surface deviations. Izv VUZ Aviats Tekh 18:58-62. [Transl Soviet Aeronautics 18(4):44-48]

Gusarov AA, Dvoretskiĭ, VM, Ivanov, MYa, Levin VA, Chernyĭ, GG (1979) Theoretical and experimental investigation of the aerodynamic characteristics of three-dimensional bodies. Izv Akad Nauk SSSR Mekh Zhidkosti i Gaza No 3, 97-102. [Transl Fluid Dynamics 14:402-406]

Sedov LI (1959) Similarity and dimensional methods in mechanics. Academic Press, New York

Tracy RR (1963) Hypersonic flow over a yawed cone. Memo 69 Calif Inst Tech Graduate Aeronautical Labs

Yahalom R (1971) An experimental investigation of supersonic flow past yawed cones. Rept AS-71-2 Univ Calif Berkeley

This article was processed using Springer-Verlag TEX Shock Waves macro package 1990.

AIAA 92-0346

Detonation Wave Propagation in Variable Density Multi-Phase Layers

S. Eidelman and X. Yang
Science Applications International Corporation
McLean, VA 22102

30th Aerospace Sciences Meeting & Exhibit

January 6-9, 1992 / Reno, NV

Detonation Wave Propagation in Variable Density Multi-Phase Layers

Shmuel Eidelman and Xiaolong Yang
Science Applications International Corporation
McLean, VA 22102

ABSTRACT

A mathematical model is presented describing a physical system of detonation waves propagating in a solid particle/air mixture with a wide range of solid phase concentrations. The mathematical model was solved numerically using the Second Order Godunov method, and numerical solutions were validated for detonation waves propagating in mixtures with concentrations of solid phase from 0.75 kg/m^3 to 1000 kg/m^3. Numerical solution was obtained for detonation waves propagating in a system consisting of layers of explosive powder with substantial variation in particle density between the layers. The study revealed a specific detonation front structure that is dependent on the thickness of the layers and their energetic content. The dynamics of lateral initiation of the adjacent layers and the structure of detonation waves in this system were investigated. Results are given for detonation of clouds having a small concentration of particles and a ground layer in which solid particle densities are three orders of magnitude larger than in the cloud.

1. INTRODUCTION

It is of considerable practical interest to study diffraction and transmission of the detonation waves into bounding layers of explosives. When combustible particles are intentionally or unintentionally dispersed into the air, the resulting mixture can be detonable. Formation of this potentially explosive dust environment and the properties of its detonation are of significant practical interest in view of its destructive or creative effects. The experimental and theoretical study of these phenomena until now has addressed only homogeneous particle/oxidizer mixtures. However, intentional or accidental processes of the explosive dust dispersion will always lead to inhomogeneous particle density distribution. Some industrial methods of explosive forming rely on detonation of explosive powder. This powder can be deposited as a thin layer over the surface area of the forming metal, with some remaining concentration in the vicinity of the layer. Also a multi-layer system can be formed from several layers of condensed explosives of different density. The structure of the detonation waves, phenomenology of its initiation, and propagation in these environments, are the main subjects of this paper.

When the detonation wave is generated in a homogeneous mixture by a "direct initiation," it starts with a strong blast wave from the initiating charge. As the blast wave decays, combustion of the reactive mixture behind its shock front starts to have a larger role in support of the shock wave motion. When the initial explosion energy exceeds some critical value, transition to steady state detonation occurs.[1-4] In explosive dust mixtures with a nonuniform distribution of particle density, the initiation dynamics is significantly more complicated. The critical initiation energy sufficient for one of the explosive particle density regions is not necessarily adequate for other regions. Also, when there is a significant variation in density between the different layers (regions) of the mixture, steady detonation in one layer can result in an overdriven detonation in an adjacent layer. Liu et al.[5] has studied experimentally a system of gaseous layers and lateral interactions for gaseous detonations. Our paper demonstrates that the phenomenology of these interactions is somewhat different from these experimental studies of multi-layer detonations in gases. This is primarily because the energy content of adjacent layers in a typical multi-gas layer experiment[5] varies by a factor of less then two, whereas the energy content in explosive dust/air mixtures can vary by several orders of magnitude.

In this paper we use detailed numerical simulation to study the initiation dynamics and propagation phenomenology for a general case of explosive dust dispersion. We will consider particle density variation from 1000 kg/m^3 in the ground layer to 0.75 kg/m^3 for the upper edges of the cloud. The effects of variation of the cloud density on detonation wave parameters will be examined for different cases of cloud particle density distribution. When possible, the results of computer simulations are validated in comparison with experimental and theoretical studies.

Section 2 of this paper describes a mathematical model that includes governing conservation equations for two phases and the constitutive laws, as well as the model for a particle gas interaction, combustion and equation-of-state for gas phase. The numerical integration technique for solving the mathematical model will is also outlined. In Section 3, we present our numerical simulation results. We first validate our model by comparing one-dimensional detonation wave simulation with available experimental results. We then give the two-dimensional simulation for detonation wave propagation in combustible particle/air mixtures with variable parti-

cle density distribution. Concluding remarks are given in Section 4.

2. THE MATHEMATICAL MODEL AND THE NUMERICAL SOLUTION

The mathematical model consists of conservation governing equations and constitutive laws that provide closure for the model. The basic formulation adopted here follows the two-phase fluid dynamics model presented in the text by Kuo[7]. The approach assumes that there are two distinct continua, one for gas and one for solid particles, each moving at its own velocity through its own control volume. The sum of these two volumes represents an average mixture volume. With these assumptions, distinct equations for continuity, momentum and energy are written for each phase. The interaction effects between the two phases are accounted as the source terms on the right hand side of the governing equation. The following is a short description of the two phase flow model used in our study, with conservation equations written in Eulerian form for two-dimensional flow in Cartesian coordinates.

<u>Conservation Equations</u>

Continuity of gaseous phase

$$\frac{\partial \rho_1}{\partial t} + \frac{\partial(\rho_1 u_g)}{\partial x} + \frac{\partial(\rho_1 v_g)}{\partial y} = \Gamma \; ; \qquad (2.1)$$

Continuity of solid particle phase

$$\frac{\partial \rho_2}{\partial t} + \frac{\partial(\rho_2 u_p)}{\partial x} + \frac{\partial(\rho_2 v_p)}{\partial y} = -\Gamma \; ; \qquad (2.2)$$

Conservation of momentum of gaseous phase in x-direction

$$\frac{\partial(\rho_1 u_g)}{\partial t} + \frac{\partial(\rho_1 u_g^2 + \phi p_g)}{\partial x} + \frac{\partial(\rho_1 u_g v_g)}{\partial y} = -F_x + \Gamma u_p \; ; \qquad (2.3)$$

Conservation of momentum of solid particle phase in y-direction

$$\frac{\partial(\rho_1 v_g)}{\partial t} + \frac{\partial(\rho_1 u_g v_g)}{\partial x} + \frac{\partial(\rho_1 v_g^2 + \phi p_g)}{\partial y} = -F_y + \Gamma v_p \; ; \qquad (2.4)$$

Conservation of momentum of solid particle phase in x-direction

$$\frac{\partial(\rho_2 u_p)}{\partial t} + \frac{\partial(\rho_2 u_p^2)}{\partial x} + \frac{\partial(\rho_2 v_p u_p)}{\partial y} = F_x - \Gamma u_p \; ; \qquad (2.5)$$

Conservation of momentum of solid particle phase in y-direction

$$\frac{\partial(\rho_2 v_p)}{\partial t} + \frac{\partial(\rho_2 u_p v_p)}{\partial x} + \frac{\partial(\rho_2 v_p^2)}{\partial y} = F_y - \Gamma v_p \; ; \qquad (2.6)$$

Conservation of energy of gas phase

$$\frac{\partial(\rho_1 E_{gT})}{\partial t} + \frac{\partial(\rho_1 u_g E_{gT} + u_g \phi p_g)}{\partial x} + \frac{\partial(\rho_1 v_g E_{gT} + v_g \phi p_g)}{\partial y}$$

$$\Gamma\Big(\frac{u_p^2 + v_p^2}{2} + Echem + C_s \bar{T}_p\Big) - \Big(F_x u_p + F_y v_p\Big) - \dot{Q} \; ; \qquad (2.7)$$

Conservation of energy of solid particle phase

$$\frac{\partial(\rho_2 E_{pT})}{\partial t} + \frac{\partial(\rho_2 E_{pt} u_p)}{\partial x} + \frac{\partial}{\partial y}(\rho_2 E_{pt} v_p) = \dot{Q} + (F_x u_p + F_y v$$

$$-\Gamma\Big(\frac{u_p^2 + v_p^2}{2} + Echem + C_s \bar{T}_p\Big) \; ; \qquad (2.8)$$

Conservation of number density of solid particle

$$\frac{\partial N_p}{\partial t} + \frac{\partial(N_p u_p)}{\partial x} + \frac{\partial(N_p v_p)}{\partial y} = 0 \, . \qquad (2.9)$$

In the above equations, $\phi = 1 - \frac{N_p M_p}{\rho_p}$, $\rho_1 = \phi\rho_g, \rho_2 = (1-\phi)\rho_p$, where N_p and M_p are the number density and mass of each particle, respectively, and ρ_g and ρ_p are the material density of gas and particle densities, respectively. u_g, v_g, p_g are gas phase x-velocity, y-velocity and pressure, respectively; $u_p, v_p, \bar{T}_p$, are x-velocity, y-velocity and average temperature of particle, respectively. C_s is specific heat of solid particle and *Echem* is chemical energy of solid phase, Γ is the rate of phase change from solid to gas and Q is heat transfer between the two phases; F_x, F_y are the drag force between the two phases in x and y directions, respectively.

Equations (2.2) and (2.7) are linked through the relation $\rho_2 = nM_p$. In the case of a reactive solid phase, M_p decreases due to combustion. The mass of a single particle at any point can be obtained from $M_p = \rho_2(x,y)/n(x,y)$, and the diameter of a particle at any spatial location is $D(x,y) = [6M_p(x,y)/\pi\rho_p]^{1/3}$. The total internal energy of gaseous phase

$$E_{gT} = E_g + \frac{1}{2}(u_g^2 + v_g^2) \quad and \quad E_g = E_g(p_g, \rho_g) \quad (2.10)$$

where $E_g(P_g, \rho_g)$ is the equation-of-state for the gas phase, which will be discussed later.

The total internal energy of solid particle phase is

$$E_{pt} = E_p + \frac{1}{2}(v_p^2 + v_p^2) \quad and \quad E_p = Echem + C_s\bar{T}_p \ . \tag{2.11}$$

In order to close the above system of conservation equations, it is necessary to define certain criteria and interaction laws between the two phases, which include mass generation rate, Γ, drag force between particles and gas, F_x, F_y and the interphase heat transfer rate $\dot{Q}$. The model for particle and gas interaction and particle combustion that results in the constitutive relation for the conservation equations, is explained in detail in the next subsection.

Model for a Particle Gas Interaction and Combustion

Presently, the physics of the energy release mechanisms in solid particles/air mixtures is not clearly understood. This can be attributed to the obvious difficulties of making a direct non-obtrusive measurement in the optically thick environment typical for this system. In the experimental and theoretical work done for the grain dust detonation conditions,[7] it was demonstrated that the volatile components released by the particle heated behind the shock front play a major role in determining the detonability limits of the mixture. Eidelman and Burcat[8] successfully applied a combination of fast evaporation and aerodynamic shattering mechanisms to simulate a two-phase detonation process.

The chemical processes of a single particle combustion, which mainly occur in the gaseous phase, are significantly faster than the physical processes of particle gasification or disintegration. Thus, in the multi-phase mixtures, the rate of energy release will be mostly determined by physics of particle disintegration. It is very difficult to describe the details of particle disintegration in the complex environment prevalent behind the shock or detonation wave. For example, Reinecke and Waldman[9] defined five different disintegration regimes for a relatively simple environment of water droplets passing through a weak shock. Fortunately, in most cases of multi-phase detonation, only the main features of the particle disintegration dynamics need to be captured to describe the phenomena. For example, Eidelman and Burcat,[10] using simple models for particle evaporation and shattering, obtained simulation results that compared very favorably with experimental data. Because of our inability to resolve the particle disintegration problem in all its complexity, the validation of the model against known experimental data is essential.

In this paper we consider solid particles consisting of explosive material. Explosive material contains fuel and oxidizer in a passive state at low temperature; however, when the temperature rises the fuel and oxidizer react, leading to detonation or combustion. The initiation of reaction for explosives will occur at relatively low temperature. For example, TNT will detonate when heated to the temperature[11] of 570°C. Only particles larger than a critical detonation size can detonate directly when initiated by a shock wave. We consider here particles smaller than 4mm in diameter that will not detonate when heated, but will burn when the temperature on the particle surface reaches a critical value. Since the heat conduction inside the explosive material is relatively slow, the process of particle heating needs to be resolved in detail. Our simulations numerically solve the temperature field in the particles at every step of numerical integration of the global conservation equations. The explosive particle combustion model examined in this paper assumes that the fraction of the particle that reaches the critical temperature will burn instantaneously.

Energy transfer by convection and conduction is simulated by solving the unsteady heat conduction equation in each computational cell at each time step. Assuming a particle's temperature T_p to be a function of time and radial position only, the unsteady heat conduction equation may be transformed to:

$$\frac{d^2 w}{dr^2} = \frac{1}{\alpha}\frac{dw}{dt} \ , \tag{2.12}$$

subject to the boundary conditions:

$$w = 0 \quad at \quad r = 0, \quad t > 0$$

$$k_s \frac{dw}{dr} = (h - \frac{1}{R})w = hRT_g \quad at \quad r = R, \quad t > 0 \tag{2.13}$$

where:

$w(r,t) = rT_p(r,t)$
r = radial position
T(r,t) = temperature
R = particle radius
T_g = temperature of surrounding gas
k_s = thermal conductivity of particle
h = convective heat transfer coefficient.

The Nusselt number, used to find h, is given by an empirical relation provided by Drake.[12] The gas viscosity is found from Sutherland's Law. The gas thermal conductivity is calculated by assuming a constant Prandtl number. Lastly, the boiling temperature at a given pressure is found from the Clapeyron-Clausius equation, assuming: 1) constant latent enthalpy of phase change, 2) the vapor obeys the ideal equation of state, and 3) the specific volume of the solid/liquid is negligible compared to that of the vapor. A critical temperature is also employed to serve as an upper limit to the boiling point, regardless of pressure.

Equation (2.13) with boundary condition (2.14) can be numerically integrated using either implicit or explicit schemes.

Since the particle radius, R, will become very small due to evaporation, the implicit Crank-Nicolson algorithm is used because of its stability properties and its second order temporal and spatial accuracy. Using the Crank-Nicolson scheme to predict the particle temperature profiles at times t_1 and t_2 permits easy calculation of the total energy exchange, Q between t_1 and t_2 due to convection and conduction.

Knowledge of the particle temperature profile also allows us to determine Γ, the rate of phase change from solid particle to gas. Once any point at a radial location $0 \leq r \leq R$ has a temperature exceeding the boiling temperature, the entire mass between r and R is transferred to the gas phase in one time step. In so doing, an energy equal to the product of the mass lost and the particle intrinsic energy is transferred by the particle to the gas.

The interphase drag force (Fx, Fy) is determined from the experimental drag for a sphere, as presented by Schlichting[13].

$$F_x = \left(\frac{\pi}{8}\right) N_p \rho_g C_D |\mathbf{V}_g - \mathbf{V}_p| (u_g - u_p) R^2 \qquad (2.14)$$

where

$$C_D = \begin{cases} \frac{24}{Re}\left(1 + \frac{Re^{2/3}}{6}\right) & \text{for Re} < 1000; \\ 0.44 & \text{for Re} > 1000, \end{cases} \qquad (2.15)$$

and $Re = \frac{2R|\mathbf{V} - V_p|}{\mu_g}$, R is radius of partricle and μg is gas viscosity at temperature of $T_{film} = \frac{1}{2}(T_g + \bar{T}_p)$. Similarly, the formulae for Fy is

$$Fy = \frac{\pi}{8} N_p \rho_g C_D |\mathbf{v}_g - \mathbf{v}_p| (v_g - v_p) R^2. \qquad (2.16)$$

Equation of State for Detonation Products

To close the system of governing equations, one needs a constitutive relation between density, pressure, temperature and energy for gas phase, which is an equation-of-state. This study uses the Becker-Kistiakowsky-Wilson (BKW) equation-of-state[14,15] that is,

$$p_g V_g / \bar{R} T_g = 1 + x e^{bx}, \qquad (2.17)$$

where V_g = volume of gas phase
p_g = pressure of gas phase
T_g = temperature of gas phase
$\bar{R}$ =- universal gas constant
$x = k / F_g (T + \Theta)^a$ $k = K \Sigma_i X_i k_i$
with empirical constants a, b, K, Θ and k_i. The constants k_i, one for each molecular species, are co-volumes. The co-volumes are multiplied by their mole fraction of species, X_i, and are added to find an effective volume for a mixture. For a particular explosive, if we know the composition of detonation products and a, b, Θ, K, and all k_i's can be found in Ref. 15.
The internal energy is determined by thermodynamics relation

$$\left(\frac{\partial E_g}{\partial V_g}\right)_T = T_g \left(\frac{\partial p_g}{\partial T_g}\right)_V - p_g. \qquad (2.18)$$

Integration of this equation for a fixed composition of the detonation products will allow us to calculate the energy of the detonation products as a function of temperature and volume. For each component, its thermodynamic properties as functions of temperature were calculated from the NASA tables compiled by Gordon and McBride[16].

The BKW equation-of-state is the most common and well calibrated of those equations-of-state used to calculate the properties of detonation products. The detailed discussion and review of the BKW equation-of-state can be found in Ref. 15.

Numerical Method of Solutions

The system of partial differential equations described in the previous paragraph is integrated numerically. The Second Order Godunov method is used for the integration of the subsystem of equations describing flow of gaseous phase material. This method is described in Ref. 17. In the following, we will elaborate only on some specifics of its application to simulations of detonation products. The subsystem of equations describing the flow of particles is integrated using a simple upwind integration. This is done because our mathematical model neglects pressure of interparticle interaction and that prevents formulation of a Second Order Godunov scheme for particles.

The physical system under study will have concentrations of solid explosive powder ranging from 1000 kg/m^3 near the ground to 0.75 kg/m^3 or less in the cloud. Detonation of this mixture will create detonation products with effective γ ranging from 3 to 1.1. To describe the flow of detonation products, we use the BKW equation-of-state described above. Since the Second Order Godunov method uses primitive variables to calculate Riemann problems at the edges of the cells, its implementation for non-ideal EOS is difficult. In our simulations, we have resolved this problem by using direct and inverse equations-of-state. After integrating a system of gas conservation laws, we use the direct BKW

equation-of-state to calculate pressure, gamma and temperature as functions of thermal energy, density, and mixture composition. After this step, we have a complete set of parameters allowing calculation of the fluxes in the Second Order Godunov method as well as interaction of the multi-phase processes. The "inverse" EOS calculates internal energy as a function of density, pressure and mixture composition. In our code we use the "inverse" EOS to calculate the fluxes of conserved variables after calculation of the flux of primitive variables.

For the multi-phase system under study, dx=dy=1mm was used to allow explicit integration of the gasdynamic and physical processes of evaporation and heat release. When a mismatch occurred between the physical and gasdynamical characteristic times, the time step was adjusted by some fraction to assure stability. However, this did not result in a significantly smaller time step as compared with that calculated by CFL criteria. For larger cell sizes, this approach is impractical. Recently we implemented a scheme in which multi-phase processes are calculated implicitly; however, this will be reported elsewhere.

The numerical method is implemented in a code named MPHASE, which is fully vectorized and supported by number of graphics and diagnostics codes.

3. RESULTS

Model Validation for One-Dimensional Detonation Wave Problem

The main advantage of our particle combustion model is its description of the phenomenology of detonation for a wide range of explosive particle sizes and densities. We will demonstrate this capability on a set of one-dimensional test problems. For these test problems, we simulated the initiation and propagation of the detonation waves in a shock tube-like setting, where the explosive particles are distributed uniformly through the shock tube volume.

Results of these simulations are summarized in Table 1, which shows detonation wave velocity, peak pressure, and peak density given as a function of the average density of the solid explosive. Here the explosive two-phase mixture is composed from RDX particles and air, where RDX particle concentration varies from 0.75 kg/m^3 to 1000 kg/m^3. This concentration variation covers the whole range of solid explosive concentrations of interest to our problem. The simulations performed with the MPHASE code were compared with the experimental results,[15,16] and the calculations presented in Ref. 19 were done with the TIGER code.

From Table I, it is clear that our simulation results compare favorably with other simulation results and experimental data. The maximum deviation between our results and referenced results is no greater than 15% for the entire range of explosives densities. Considering that our results were obtained with a single model for particle combustion applied to the extreme range of densities. our model gives an excellent prediction of the detonation wave parameters.

Two-Dimensional Simulation Results

In our two-dimensional simulations, we first study the dynamic of the lateral initiation in a simple system formed by two layers of explosive with different concentrations of the explosive powder in the layers. These layers of explosive will be considered confined in a rectangular shock tube with rigid walls. The schematics of the set up for a typical simulation of this type are shown in Figure 1. The detonation wave is initiated in the lower layer, and its propagation though the shock tube causes lateral initiation of the adjacent layer. In one of the test cases, both layers are initiated simultaneously with a planar front.

First we simulated initiation and propagation of the detonation in a system of two layers of detonable RDX powder/air mixture contained in a rectangular channel 4 cm wide and 35 cm long. The lower layer has an RDX powder concentration of 800 $\frac{kg}{M^3}$ and occupies half of the channel width, and the upper layer of the channel has a mixture concentration of 200 $\frac{kg}{M^3}$. Detonation is initiated in the lower layer by a planar front that is propagating from left to right. In Figures 2a:2f, results of this simulation are shown in the form of pressure contours on a logarithmic scale in MPa for a sequence of time frames. In these figures, we can follow the evolution of the lateral initiation and formation of the detonation wave structure in this system.

In Figure 2a, contour plots are shown at time t=0, which corresponds to the beginning of the simulation and depicts initial conditions of the planar wave in the lower layer. This initial wave causes lateral initiation of the upper layer through an oblique detonation front shown in Figure 2b at $t=9\times10^{-6}$ sec. The oblique front reflects from the upper wall of the channel, and in Figure 2c we observe that the wave pattern indicates it is a single Mach reflection. The Mach stem is very short at this point. In Figure 2d, the pressure contours are shown at the time $t=31\times10^{-6}$ sec. Here the Mach stem is clearly visible and the reflected shock has reached the lower wall of the channel. The Mach stem will continue to grow and the triple point will propagate towards the high density layer. In Figure 2e, the simulation results are shown at $t=52\times10^{-6}$ sec when detonation wave complex has reached steady state propagation regime. The triple point has reached the interface between the two layer and is unable to continue propagation downwards due to the high level of pressure and density in the lower layer. Also at this stage of the detonation wave propagation, the reflected shock has reached the upper wall

of the channel. In Figure 2f, the simulation results are shown at $t=64 \times 10^{-6}$ sec. Here the structure of the detonation front is basically unchanged from the previous picture, except for an additional reflection from the upper wall of the channel. The detonation wave parameters are also unchanged from the previous time frame, indicating that the detonation wave in this two layer system has reached steady state.

To validate that the detonation waves complex observed in above reported simulation is not a function of the initial conditions, we simulated a test case in which all problem parameters, except the initiation wave, are the same as in the previous case. The initiation is done by a single planar wave that starts propagating simultaneously in both layers of the explosive. In Figures 3a:3e, results for this simulation are shown in the form of pressure contours for a sequence of time frames. The initial conditions are shown in Figure 3a. Here we can observe a planar front impinging simultaneously on both layers of explosive in the channel. At first, this front propagates some distance planarly, as observed in Figure 3b. However, a significant difference in the explosive powder density quickly leads to formation of the oblique front in the upper layer, as shown in Figure 3c. As in the previous case, the oblique front reflects from the upper wall in the single Mach reflection shown in Figure 3d. And as in the previous case, the triple point of the Mach stem propagates downward to the interfaces between the layers to form the stable wave pattern shown in Figure 3e. The parameters of the detonation waves and the structure of the detonation wave complex are identical to those observed in the previous case, which proves that it is not a function of the initial conditions, but physical conditions of the layers.

We studied the effects of the channel walls using a system that included a 2cm thick lower layer of high density (800 $\frac{kg}{M^3}$) RDX powder and a 10cm thick upper layer of low density (200 $\frac{kg}{M^3}$) RDX powder. The results of this simulation are shown as pressure contours on a logarithmic scale in Figures 4a:4d. Figure 4a shows the initial conditions. In Figure 4b, we can see at the time $t = 25 \times 10^{-6}$ a planar detonation wave is propagating through the lower layer and an oblique wave is propagating through the upper layer. In Figure 4c, the detonation wave is shown at the time $t=41 \times 10^{-6}$ from the initiation. Here the oblique wave is reflecting from the upper wall; however, it is distinct from the previous cases because only a regular reflection pattern is formed. This is due to the shallow angle of incidence of the detonation wave, that corresponds to the large wedge angles in classical reflection problems. Figure 4d shows the results of the simulation at $t = 52 \times 10^{-6}$. Here we can observe the same regular reflection pattern as in the previous stage; however, the incidence angle of the oblique wave in the upper layer is increasing. Thus, if this trend continues, later in the detonation wave evolution we will see the formation of the Mach reflection pattern, as we have in previous cases.

We have also examined propagation of the detonation wave in the system shown in Figure 5 that corresponds to the situation where the upper layer is not confined by the channel wall. Here the computational domain is 25cm × 25cm in size. The explosive powder density is distributed according to the 4th power law of vertical distance, starting from the ground where the density is 860 kg/m^3, to 1.2cm, where the density is 0.75 kg/m^3. From this point to 25cm height, the density is constant and equal to 0.75 kg/m^3. The density distribution in the direction of the "x" axis is uniform. The boundary conditions for the computational domain shown in Figure 5 are specified as follows: solid wall along the "x" axis; symmetry conditions along the "y" axis; supersonic outflow for upper boundary and at the right of the computational domain. The mixture consists of RDX powder and air at ambient conditions and it is assumed to be quiescent at the time of initiation.

The simulation starts at t=0 when the mixture is initiated at the lower left corner of the computational domain, as shown in Figure 5. The energy released by the initiating explosion leads to formation of the detonation wave propagating through the multi-phase media. Figure 6a shows pressure contours for the propagating detonation wave at the time of $t=12 \times 10^{-6}$ msec after initiation. Here the pressure contour levels are shown on logarithmic scale in MPa. The maximum pressure value of 7940 MPa is observed in the layer of condensed explosive located near the ground. The pressure in the layer is two to three orders of magnitude higher than pressure behind the detonation wave in the 0.75 kg/m^3 RDX cloud and air, which is located above the distance of 1.2cm from the ground. Figure 6a demonstrates that the detonation wave in the cloud is overdriven, since the pressure behind the shock continuously rises and reaches its maximum in the layer. From this figure, we also observe that the overdriven wave propagates faster in the cloud than in the layer. This is explained by the fact that it is easier to compress air that is very lightly loaded with particles and located above the ground layer, than it is to compress air heavily loaded with a particle mixture near the ground. It is interesting to note a discontinuous pressure change between the yellow contours and the light blue and green contours behind the detonation front. This discontinuity is over-emphasized by our presentation of contour lines on the logarithmic scale; however, further examination of our simulation results indicates this feature is real and is similar in nature to barrel shocks observed for strong jets. It is different in nature from the triple shock structures described above.

In Figure 6b, gas phase density contours are shown for the time $t = 12 \times 10^{-6}$ sec. Here the contour lines are distributed on logarithmic scale. The main features of the shock wave structure are very similar to those observed in the pressure contours figure. Here we see that a jet of high density gases reflects from the center of symmetry axis, creating a contact discontinuity that we will observe at a later time. The barrel shock is clearly visible in this figure. In Figure 6c, the particle density contour plots are shown for $t=12 \times 10^{-6}$ sec. The contour levels in this figure are given on the logarithmic scale and the initial deposition of the explosive material in the ground layer of the computational domain can be clearly observed. The black contour lines delineate the beginning and the end of the reaction zone in the cloud. To the left of these contours lies an area with combustion products and to the right unburned particles in the cloud. Here we can see that the reaction zone length is of the order of 1cm.

Figure 6d shows pressure contours for the same simulation for the time $t = 55 \times 10^{-6}$ sec, just before the detonation wave leaves the computational domain. In this figure, we see that the global structure of the wave did change slightly from Figure 6a. We observe that the barrel shock wave is fully developed and has a half ellipse shape. The detonation wave in the cloud is still overdriven; however, part of the shock wave front that propagates vertically weakened as it got further away from the detonation front in the layer. In Figure 6e, gas temperature contours are shown at $t = 55 \times 10^{-6}$ sec. In this case, it is interesting to note that the highest temperatures are observed behind the front of the overdriven detonation wave in the cloud, in the immediate vicinity of the upper strata of the layer. Very high temperatures in this region can be explained by the high pressure generated by the detonation of the explosive material in the layer and by relatively low density of strata of the cloud in the immediate vicinity to the layer. Here, as in the pressure contours graph, the area of barrel shock can be clearly identified.

We also observe in Figure 6 a clear development of two detonation fronts, one moving vertically in the cloud and another moving horizontally in the layer. Because the energy density of the explosive powder in the layer is about three orders of magnitude larger than that in the cloud, the vertical parts of the front represent overdriven detonation waves in the cloud. Even though the vertical front has slowed down compared with the horizontal front, its speed and parameters far exceed those typical for detonation waves in a cloud. In fact, the self-sustained detonation regime in the cloud will develop at the distance of about three meters from the layer. The area of the front close to the detonation wave in the layer will remain hot and overdriven, since it is located very close to detonation front in the layer. In Figure 6f, particle density contours are shown on a logarithmic scale. We can clearly observe the reaction zone delineated by black contour lines. In this case, the reaction zone length in the cloud is about 1cm. Consistent with the gradual transition from overdriven to self-sustained detonation, the reaction zone length is larger for the vertical part of the detonation front. The detonation wave velocity observed in our simulation is approximately 4048 m/sec, which is significantly lower than the detonation wave velocity observed in RDX with a density of 860 kg/m^3 (see Table 1), which is the highest density in the ground layer. This can be explained by the high gradient of particle density distribution in the layer, where the density drops rapidly from 860 kg/m^3 at the bottom of the layer to 0.75 kg/m^3 at the top strata of the layer at 12 mm above the ground.

4. CONCLUSIONS

We have presented a mathematical model and numerical solution for the simulation of initiation and propagation of the detonation waves in multi-phase mixtures consisting of solid combustible particles and gas. Using this model, we studied detonations in mixtures of solid RDX particles and air for the purpose of examining the effects of wide variation in particle density distribution on the dynamics and structure of detonation waves. We considered a physical system of layers of explosive RDX powder confined in a channel and studied initiation and propagation of the detonation waves in this system. This study revealed a specific structure of the detonation front that is dependent on the thickness of the layers and their energetic content. We showed that for the system consisting of two layers of the same thickness but of vastly different powder density, a Mach stem reflection occurs that propagates to the interface between the layers and helps create a stable detonation front. However, formation of the Mach stem reflection will be a strong function of the relative thickness of the layer; in one of the simulated examples, only a regular reflection would form in the simulation time frame.

For the system consisting of a solid particle cloud in air and a layer of high particle density near the ground, our simulations have revealed a specific detonation front shape with a characteristic precursor of the blast front in the strata immediately above the layer. This feature of the detonation front can be explained by the fact that the energy released in the detonation wave in the ground layer produces a faster shock wave in the dilute cloud than in these heavily loaded with solid particles stratums of the ground layer. However, these structures were not observed experimentally, and more studies are needed to examine their parameters.

The maximum pressure affecting the ground was di-

rectly related to the maximum particle density in the lower strata of the layer. However, the detonation front velocity for the fourth power distribution case was considerably lower than calculated for a one-dimensional case with 860 kg/m^3 particle density, reflecting the significant effect of two-dimensional expansion. Existence of the high density strata at the bottom of the ground layer in the fourth power case significantly increased the maximum pressure at the ground, and produced higher detonation wave velocity.

Using a variable density layer, one can reach a combination of pressure and velocity conditions outside of Chapmen-Jougett limitations. The range of conditions that can be obtained in the variable density system and its parametrics of that system needs a more systematic study. In this article, we introduced only the mathematical formulation and numerical simulation method validated for the range of conditions of interest. In addition, we have given some examples of its application for two-dimensional simulations. However, this methodology should be linked to an experimental study for a more in-depth analysis of the phenomenology discussed here.

REFERENCES

1. Eidelman, S., Timnat, Y.M., and Burcat, A., (1976). "The Problem of a Strong Point Explosion in a Combustible Medium," 6th Symp. on Detonation, Coronado, CA, Office of Naval Research, 590.
2. Burcat, A., Eidelman, S., and Manheimer-Timnat, Y., (1978). "The Evolution of a Shock Wave Generated by a Point Explosion in a Combustible Medium," Symp. of High Dynamic Pressures (H.D.P.), Paris, 347.
3. Oved, Y., Eidelman, S., and Burcat, A., (1978). "The Propagation of Blasts from Solid Explosives to Two-Phase Medium," Propellants and Explosives, 3, 105.
4. Eidelman, S., and Burcat, A., (1980). "The Evolution of a Detonation Wave in a Cloud of Fuel Droplets; Part I, Influence of the Igniting Explosion," AIAA Journal, 18, 1103.
5. Liu, J.C., Kauffman, C.W., and Sichel, M., (1990). "The Lateral Interaction of Detonating and Detonable Mixtures," (Private communication).
6. Kuo, K., (1990). "Principles of Combustion," John Wiley and Sons, Inc.
7. Kauffman, C.W., et al., (1979). "Shock Wave Initiated Combustion of Grain Dust," Symposium on Grain Dust, Manhattan, KS.
8. Eidelman, S., and Burcat, A., (1981). "Numerical Solution of a Non-Steady Blast Wave Propagation in Two-Phase ('Separated Flow') Reactive Medium," J. Comput. Physics. 39, 456.
9. Reinecke, W.G., and Waldman, G.D., (1975). "Shock Layer Shattering of Cloud Drops in Reentry Flight," AIAA Paper 75-152.
10. Eidelman, S., and Burcat, A., (1980). "The Mechanism of Detonation Wave Enhancement in a Two-Phase Combustible Medium," 18th Symposium on Combustion, The Combustion Institute, Waterloo, Ontario, Canada.
11. "Engineering Design Handbook, Explosives Series, Properties of Explosives of Military Interest," AMC Pamphlet, AMCP 706-7177, 1971.
12. Drake, R.M., Jr., (1961). "Discussions on G.C. Vliet and G. Leppert: Forced Convection Heat Transfer from an Isothermal Sphere to Water," Journal of Heat Transfer, 83, 170.
13. Schlichting, H., (1983). "Bounday Layer Theory," 7th ed. McGraw-Hill.
14. Cowan, R.D., and Fickett, W., (1956). "Calculation of the Detonation Products of Solid Explosives with the Kistiakowsky-Wilson Equation of State," Journal of Chemical Physics, 24, 932.
15. Mader, C.L., (1979). "Numerical Modeling of Detonation," University of California Press, Ltd. London, England.
16. Gordon, S., and McBride, B.J., "Computer Program for Calculations of Complex Chemical Equilibrium Compositions, Rocket Performance, Incident and Reflected Shocks and C-J Detonations," NASA SP-273, 1976 Revision.
17. Eidelman, S., Collela, P., and Shreeve, R.P., (1984). "Application of the Godunov Method and Its Second Order Extension to Cascade Flow Modelling," AIAA Journal, 22, 10.
18. Stanukovitch, K.P., (1975). "Physics of Explosion" (in Russian), Nauka.
19. Wiedermann, A., (1990). "An Evaluation of Bimodal Layer Loading Effects," IITRI Report, February.

D[m/sec] - Detonation wave velocity,
P_{CJ}[Pa] - Pressure at Chapman-Jouguet Point
P_p[Pa] - Peak pressure; ρ_p[kg/m^3] - Peak density

RDX Density (kg/m^3)	Parameters	Present Calculation	Expt'l Ref. 1	Tiger Calculation Ref. 2	BKW Calculation Ref. 1	Soviet Experiments Ref. 3
1000 kg/m^3	D	6155	5981		6128	
	P_{CJ}	1.220×10^{10}			1.08×10^{10}	1.00×10^{10}
	P_p	2.57×10^{10}				
	ρ_p	1936				
860 kg/m^3	D	6031		5900		
	P_{CJ}	0.986×10^{10}		0.88×10^{10}		0.82×10^{10}
	P_p	1.95×10^{10}				
	ρ_p	1722				
466 kg/m^3	D	4800		4500		
	P_{CJ}	0.379×10^{10}		0.30×10^{10}	0.3×10^{10}	
	P_p	0.625×10^{10}				
	ρ_p	924				
250 kg/m^3	D	4049		3600		
	P_{CJ}	0.2478×10^{10}		0.13×10^{10}		
	P_p	0.4538×10^{10}				
	ρ_p	552				
100 kg/m^3	D	3495				
	P_{CJ}	0.5013×10^{9}				
	P_p	0.7658×10^{9}				
	ρ_p	220				
0.75 kg/m^3	D	1622	1410*	1870*		
	P_{CJ}	0.25×10^{7}	0.284×10^{7}*	0.26×10^{7}*		
	P_p	0.484×10^{7}				
	ρ_p	8				

Ref. 1 - Mader, C., "Numerical Modeling of Detonation," (University of California Press, Ltd., 1979), p. 47.
Ref. 2 - Wiedermann, A., "An Evaluation of Bimodal Layer Loading Effects," IITRI Report, Feb., 1990.
Ref. 3 - Stanukovitch, K.P., "Physics of Explosion" (in Russian), Nauka, 1975.

Table 1. One Dimensional Validation Result.

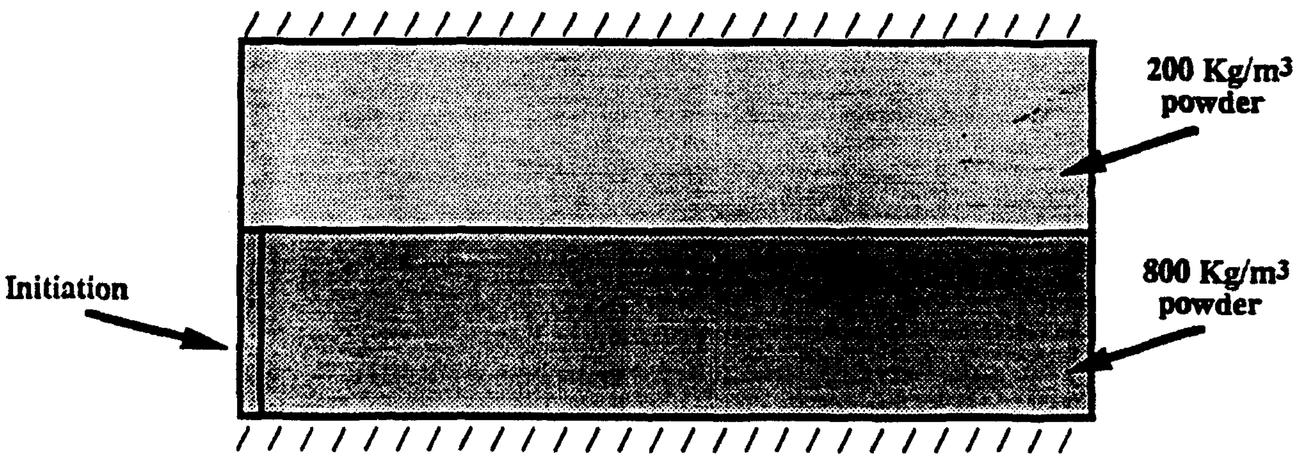


Figure 1. Setup for the two-layer detonation simulation problem.

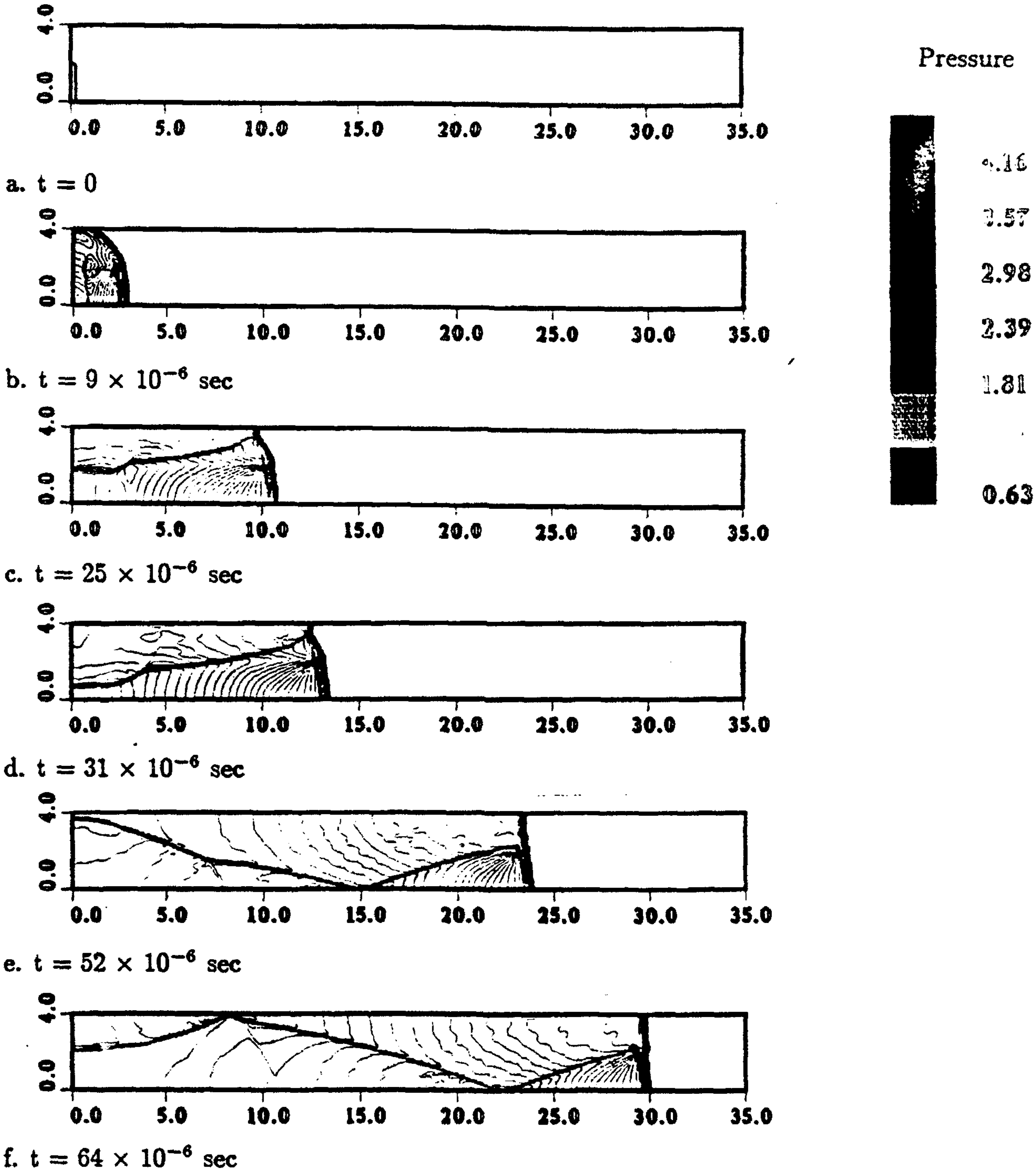


Figure 2. Initiation and propagation of the detonation wave in a two layers system. Only lower layer is initiated. Pressure contours.

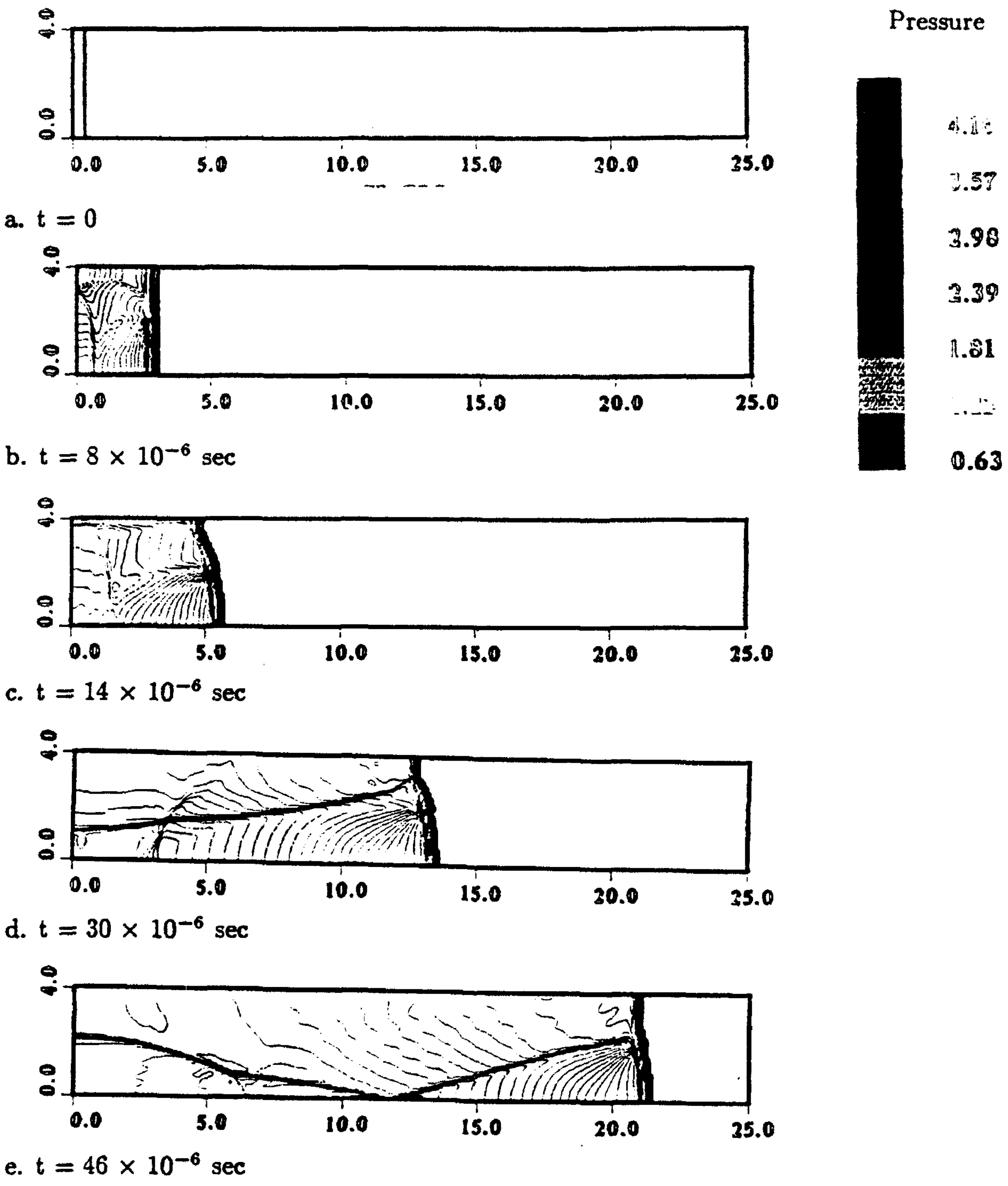


Figure 3. Initiation and propagation of the detonation wave in a two layers system. Both layers are initiated. Pressure contours.

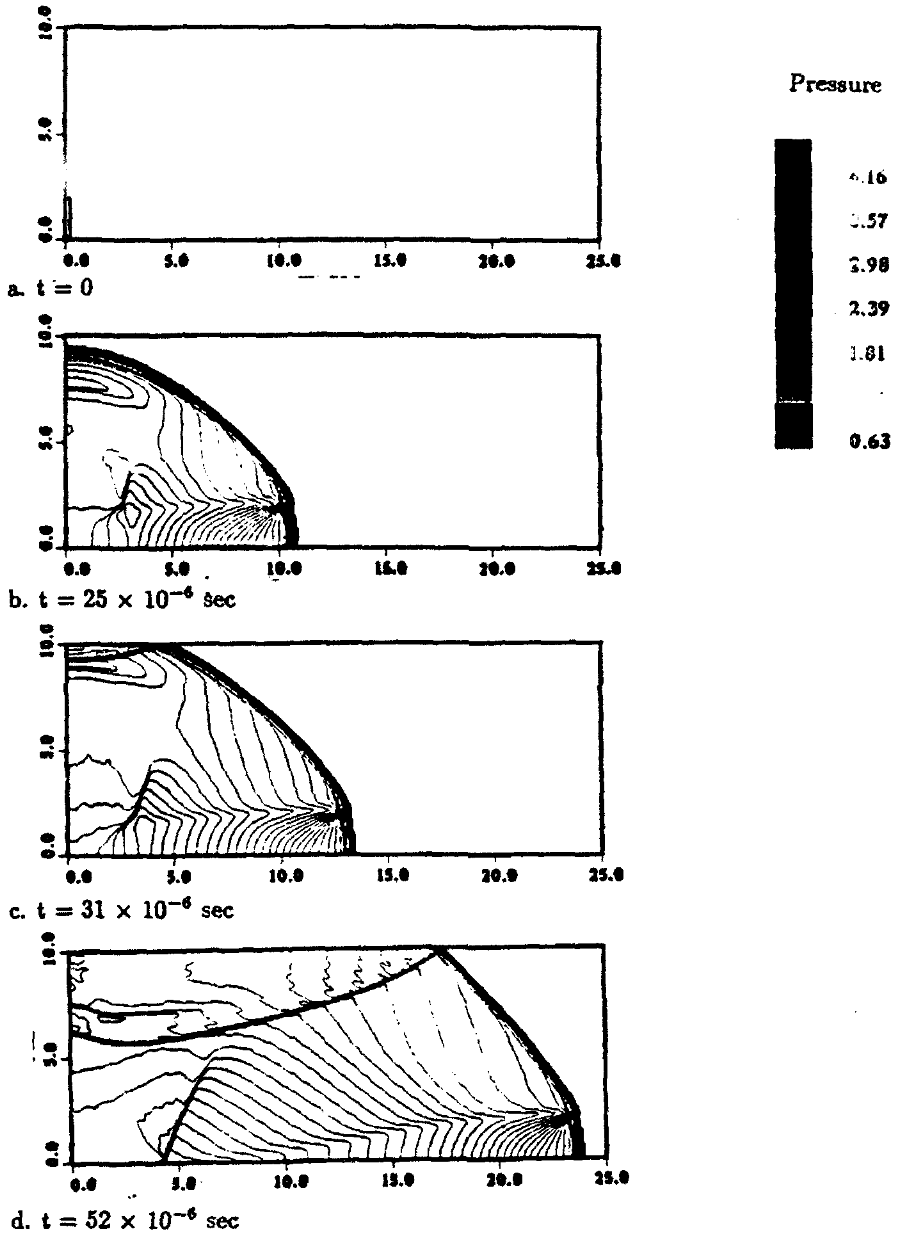


Figure 4. Propagation of the detonation wave in a system with different thickness of explosive layers. Pressure contours.

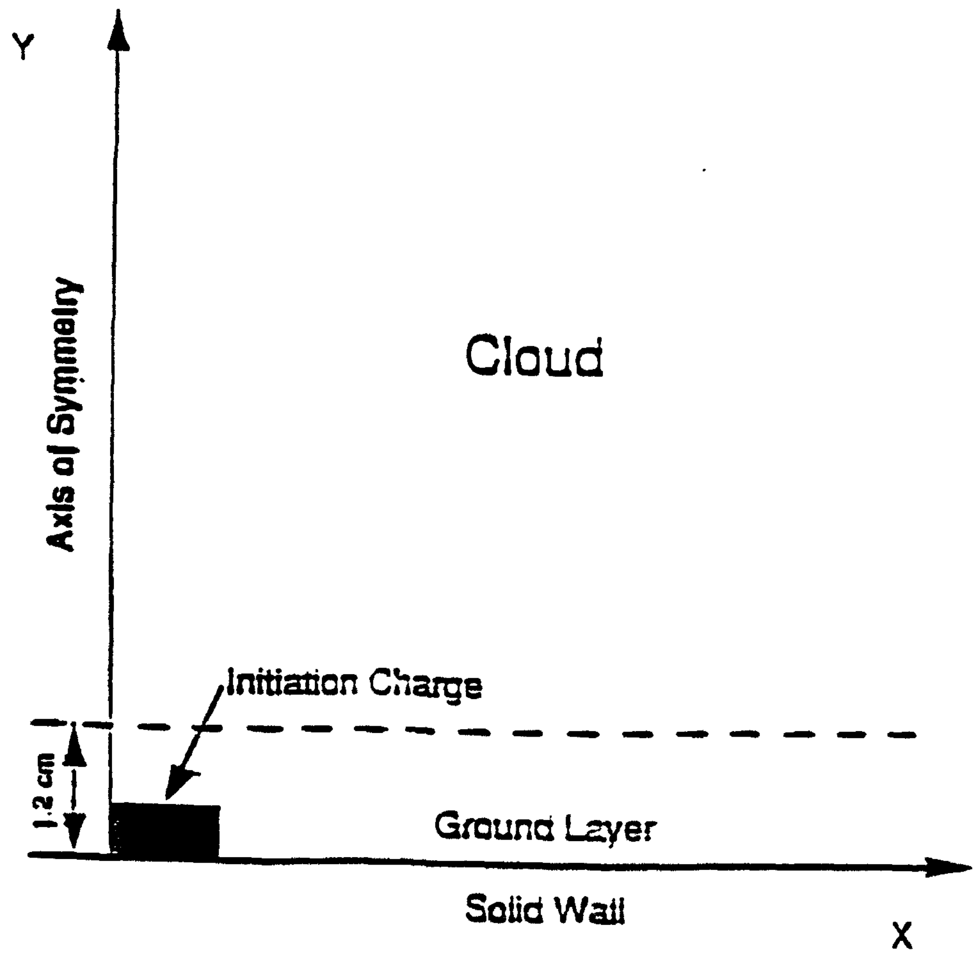


Figure 5. Computational domain and boundary conditions.

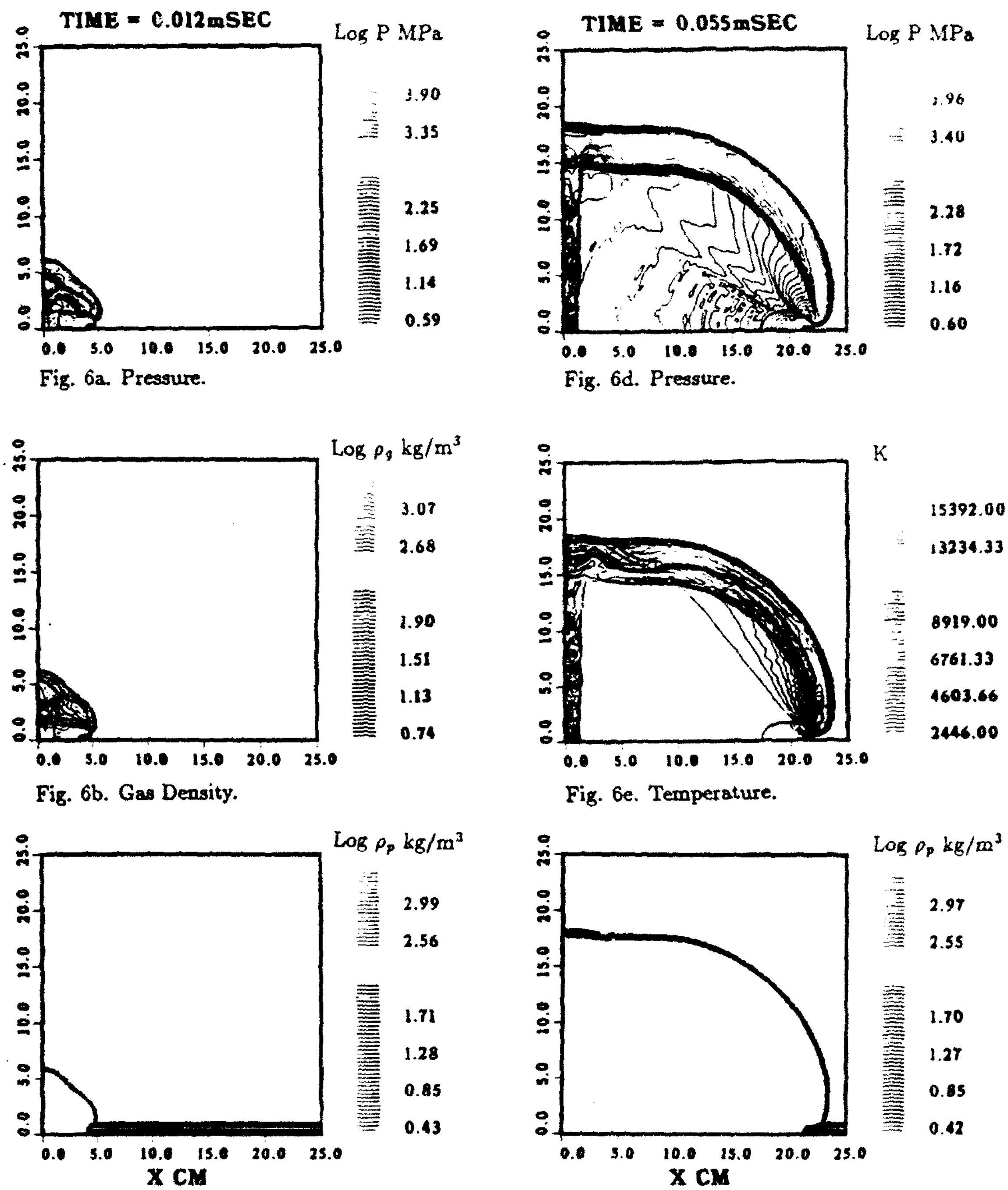


Fig. 6a. Pressure.

Fig. 6d. Pressure.

Fig. 6b. Gas Density.

Fig. 6e. Temperature.

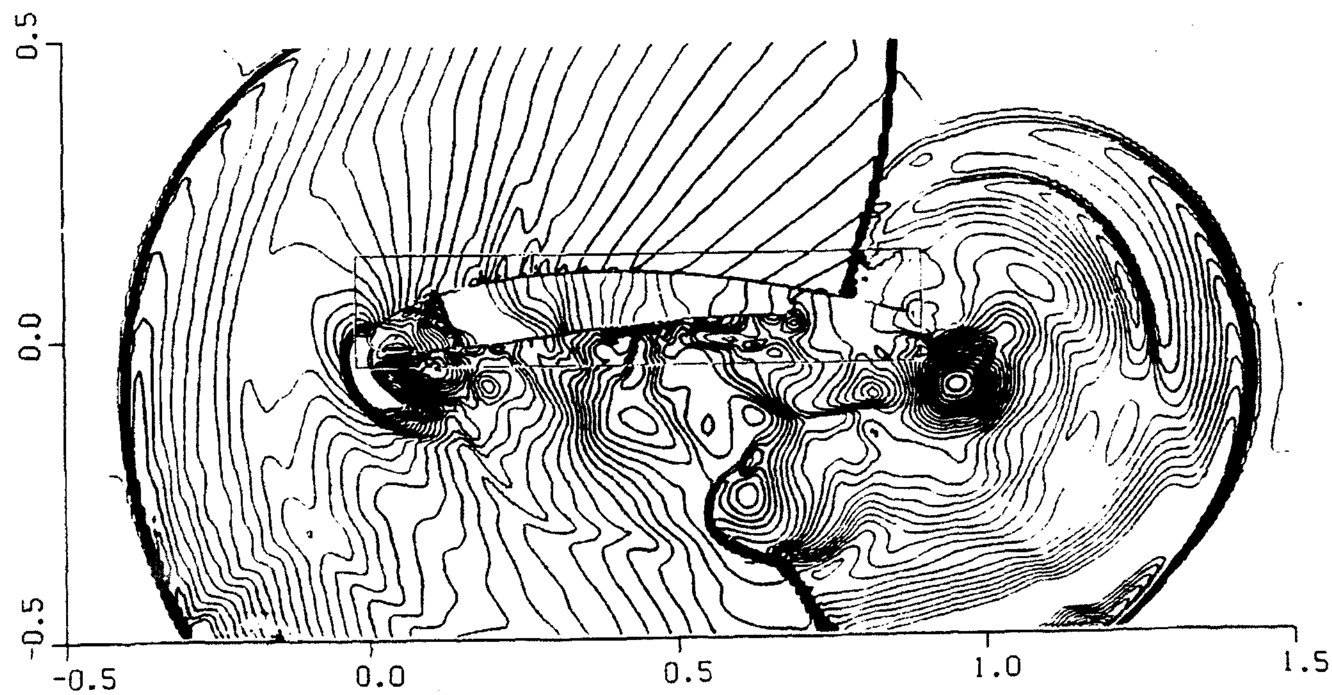
Fig. 6c. Particle Density.

Fig. 6f. Particle Density.

Figure 6. Fourth power layer distribution. Maximum density in the layer 800 kg/m^3. Density in the cloud 0.75 kg/m^3. Time 0.012 msec and 0.055 msec after initiation.

AIAA 92-0392
A Parametric Study of the Air-Breathing Pulsed Detonation Engine

S. Eidelman, I. Lottati and W. Grossmann
Science Applications International Corporation
McLean, VA 22102



30th Aerospace Sciences Meeting & Exhibit

January 6-9, 1992 / Reno, NV

A PARAMETRIC STUDY OF AIRBREATHING PULSED DETONATION ENGINE

S. Eidelman, I. Lottati, and W. Grossmann
Applied Physics Operation
Science Applications International Corporation
1710 Goodridge Drive, McLean, Virginia 22102

Abstract

The airbreathing Pulsed Detonation Engine (PDE) is analyzed by direct simulations of its cycle using Computational Fluid Dynamics. We describe a new CFD methodology of composite structured/unstructured grids, which is used for detailed analysis of the PDE performance. This performance is analyzed for a unique engine geometry in which the PDE is located in a wing section. Examination of the key processes in the PDE device shows that the largest portion of its thrust is produced during the very short time interval when the detonation wave reflects from the thrust wall, and that detonation cycle frequency up to 200Hz is feasible. We conclude that the PDE type devices can compete with small diameter turbojet engines in performance characteristics while surpassing them in simplicity of design, flexibility of geometrical configuration, and price.

1. Introduction

Our first reports on the airbreathing Pulsed Detonation Engine (PDE) concept[1-5] described a systematic series of parametric studies of the PDE via computational fluid dynamics (CFD). They also detailed an analysis of engine performance over a wide range of flight regimes, including subsonic and supersonic flows and physical geometries with various nozzle and air inlets. Additionally, static table top experiments[1] demonstrated that the principle of pulsed or repetitive detonation can be successfully applied. To date, our results indicate that practical engines for certain vehicles can be conceptualized and designed with the information that has already been generated from the studies. Specifically, our studies have shown that the PDE is an excellent candidate for the primary propulsion source for small aerodynamic vehicles that operate over the flight envelope, $0.2<M<2.0$. Further, our analysis of the simulation results indicates that the PDE is a high thrust-to-weight ratio device. The predicted performance places the PDE propulsion concept in a strongly competitive position compared with present day small turbojets. The PDE concept has the added attractiveness of rapid variable thrust control, no moving parts and the potential for low cost manufacturing. The PDE concept is scalable over a wide range of engine sizes and thrust levels.[4] For example, it is theoretically possible to produce PDE engines on the order of one to several inches in diameter and thrusts on the order of pounds, as well as devices that provide thousands of pounds thrust. One of the unique features of the PDE that will be explored in this paper is its geometric flexibility. All the configurations of the engine that we have examined in previous papers had an axisymmetric geometry. However, the PDE concept allows a tremendous flexibility in engine geometry. In this paper we will investigate the possibility of fitting a PDE det onation chamber into a section of a conventional wing One of the obvious advantages of this design is reductior of the drag and weight penalty; other advantages can b associated with stealth quality of the Wing-PDE design

The parametric studies to date were made possi ble by the development of a new generation of CFD tools. These tools have allowed us to accurately sim ulate the details of the complex nonlinear time depen dent processes. In this article, we used a new algorithr implemented on a composite structured/unstructurec grid. This algorithm combines the flexibility of describ ing complicated geometries characteristic of the unstructured triangular grids with the computational efficiency of the structured grids. A brief description of the CFD methods employed in our studies is given in Section 3.

2. The Pulsed Detonation Engine Concept

A detonation process, due to the very high rate of reaction, leads to a propulsion concept in which the constant volume process can be fully realized. In detonative combustion, the strong shock wave, which is part of the detonation wave, acts like a valve between the detonation products and fresh charge. The speed of the detonation wave is about two orders of magnitude higher than the speed of a typical deflagration. This allows the design of propulsion engines with a very high power density. Each detonation has to be initiated separately by a fully controlled ignition device with a wide range of variable cycle frequencies. A physical restriction dictating the range of detonation frequency arises from the rate at which the fuel/air mixture can be introduced into the detonation chamber. This also means that a device based on a detonative combustion cycle can be scaled and its operating

parameters can be modified for a range of required output conditions.

There have been numerous attempts to take advantage of detonative combustion for engine applications,[6,7,8] the most recent and successful which was carried out at the Naval Postgraduate School[1] (NPS) by Helman et al. During this study, several fundamentally new elements were introduced to the concept that distinguished the NPS research device from previous studies. First, it is important to note that the NPS experimental apparatus was the first successful self- aspirating air breathing detonation device. Intermittent detonation frequencies of 25 Hz were obtained, which was in phase with the fuel mixture injection through the timed fuel valve opening and spark ignition. The feasibility of intermittent injection was established. Pressure measurements showed conclusively that a detonation process occurred at the frequency chosen for fuel injection. Further, self- aspiration was shown to be effective. Finally, the effectiveness of a primary detonation as a driver for the main detonation was clearly demonstrated. Although the NPS studies were abbreviated, many of the technical issues considered to be essential for efficient intermittent detonation propulsion were addressed with positive results.

The generic device we considered in our previous studies[2-5] is a small engine shown in Figure 1, which is a schematic of the basic detonation chamber attached to the aft end of a generic aerodynamic vehicle. In the current study, we considered a Wing-PDE configuration that will be described below; however, for the sake of simplicity we will describe the basics of the PDE concept using the illustration in Figure 1. For the engine configuration shown in this figure, the combustible gas mixture is injected at the closed end of the detonation chamber and a detonation wave, initiated at the aft end of the detonation chamber, propagates through the mixture. The main portion of the thrust is produced by the detonation wave in a very short period of time as it impinges on the thrust wall. After the detonation wave has reflected from the thrust wall, the detonation products will vent from the volume of the detonation chamber through the open aft end of the chamber and air inlets shown in Figure 1. Then the chamber volume will be filled with the fresh combustible gas mixture and the process will be repeated with the frequency of 100 to 200Hz. A key issue in the pulsed detonation engine concept is the design of the main detonation chamber. The detonation chamber geometry determines the propulsion efficiency and the duration of the cycle (frequency of detonations). Since the fresh charge for the generic engine is supplied from the external flow field, the efficiency of the engine depends on the interaction of the surrounding flow with the internal flow dynamics. The range of the physical processes requiring simulation in order to model the complex flow phenomena associated with the detonation engine performance is very broad. These processes include 1) initiation and propagation of the detonation wave inside the chamber; 2) expansion of the detonation products from the chamber into the air stream around the chamber at flight Mach numbers; 3) fresh air intake from the surrounding air into the chamber; 4) the flow pattern inside the chamber during post-exhaust pressure buildup which determines the strategy for mixing the next detonation charge; and 5) strong mutual interaction between the flow inside the chamber and surrounding the engine.

All of these processes are interdependent, and interaction and timing are crucial to engine efficiency. Thus unlike simulations of steady state engines, the phenomena described above cannot be evaluated independently. The need to resolve the flow regime inside the chamber and account for nozzles, air inlets, etc., and at the same time resolve the flow outside and surrounding the engine where the flow regime varies from high subsonic, locally transonic and supersonic, makes it a challenging computational problem.

The single most important issue is to determine the timing of the air intake and mixing of the fresh charge leading to repetitive detonations. It is sufficient to assume inviscid flow for the purpose of simulating the expansion of the detonation products and fresh air intake. This assumption makes the numerical simulation of the PDE flow phenomena somewhat easier than using a fully viscous flow model. For the size of the generic device studied in this work, the effects of viscous boundary layers are negligible, with the exception of possible boundary layer effects on the valve and inlet geometries discussed subsequently.

3. Computational Method Used in the Study

The basic computational tool used for our studies is the AUGUST (Adaptive Unstructured Godunov Upwind Second Order on Triangular Grids) code, described in detail by Lottati et al.[9,10] This code provides a method for solving the Euler equations of gasdynamics on unstructured grids with arbitrary connectivity. The formulation is based on a second order Godunov method.[11] For the current study, the AUGUST code has been implemented on a composite structured/unstructured grid. The combined structured/unstructured method is a much more efficient approach to domain decomposition than the separate application of each method. In the following discussion, we show that the results of applying this technique to the complex problem of the external/internal reactive flow typical for the PDE engine show complex wave patterns propagating seamlessly through interfaces between structured/unstructured grids without reflections or distortions. This new approach provides ultimate flexibility

in domain decomposition with maximum code efficiency.

Introduction

Structured rectangular grids allow the construction of numerical algorithms that perform an efficient and accurate integration of fluid conservation equations. The efficiency of these schemes results from the extremely low storage overhead needed for domain decomposition and the efficient and compact indexing that also defines domain connectivity. These two factors allow code construction based on a structured domain decomposition that can be highly vectorized and parallelized. Integration in physical space on orthogonal and uniform grids produces the highest possible accuracy of the numerical algorithms. The disadvantage of structured rectangular grids is that they cannot be used for decomposition of computational domains with complex geometries.

The early developers of computational methods realized that, for many important applications of Computational Fluid Dynamics (CFD), it is unacceptable to describe curved boundaries of the computational domain using the stair-step approximation available with the rectangular domain decomposition technique. The techniques of boundary-fitted coordinates were developed to overcome this difficulty. With these techniques, the computational domain is decomposed on quadrilaterals that can be fitted to the curved domain. The solution is then obtained in the physical space using the geometrical information defining the quadrilaterals, or in the computational coordinate system that is obtained by transformation of the original domain into a rectangular domain. The advantage of this technique is that it employs the same indexing method as the rectangular structured domain decomposition methods that also serve to define domain connectivity. The boundary fitted coordinated approach leads to efficient codes, with approximately a 4:1 penalty in terms of memory requirement per cell as compared with rectangular domain decomposition. However, this approach is somewhat restricted in its domain decomposition capability, since distortion or large size variations of the quadrilaterals in one region of the domain lead to unwanted distortions or increased resolution in other parts of the domain. An example of this is the case of structured body fitted coordinates that are used for simulations of flows over a profile with sharp trailing edges. In this case, increased resolution in the vicinity of the trailing edge leads to increased resolution in the whole row of elements connected to the trailing edge elements.

The most effective methods of domain decomposition developed to overcome this disadvantage are those using unstructured triangular grids. These methods were developed to cope with very complex computational domains. The unstructured grid method, while efficient and powerful in domain decomposition, results in codes that must store large quantities of information defining the grid geometry and connectivity, and have large computational and storage overheads. As a rule, an unstructured grid code requires greater storage by a factor of 10, and will run about 20 times slower when compared on a per cell per iteration basis with a structured rectangular code.

Unstructured grid methods are used to their best advantage when combined with grid adaptivity. This feature usually allows dynamic decomposition of the computational domain subregions, thus leading to an order of magnitude reduction in the number of cells for some problems, as compared to the unstructured grid without this adaptive capability. However, this advantage is highly dependent on the problem solved. Adaptive unstructured grids have an advantage over the unadaptive unstructured domain decomposition if the area of high resolution domain decomposition is less than one tenth of the global area of the computational domain. This explains the fact that while the adaptive unstructured method may be extremely effective for solutions with multiple shock waves in complex geometries, it becomes extremely inefficient when high resolution is needed in a substantial area of the computational domain.

Our approach to domain decomposition combines the structured and unstructured methods for achieving better efficiency and accuracy. Using this method, structured rectangular grids are used to cover most of the computational domain, and unstructured triangular grids are used only to patch between the rectangular grids (Figure 2), or to conform to the curved boundaries of the computational domain (Figure 3). In these figures, an unstructured triangular grid is used to decompose the regions of the computational domain that have a simple geometry.

Our paper will illustrate the performance gains achieved from the use of this composite grid decomposition approach. We apply the Second Order Godunov method[11] to solve the Euler equations on both structured and unstructured sections of the grid.

Mathematical Model and Integration Algorithm

We consider a system of two-dimensional Euler equations written in conservation law form as:

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = 0 \tag{1}$$

where

$$U = \begin{vmatrix} \rho \\ \rho u \\ \rho v \\ \rho e \end{vmatrix}, \quad F = \begin{vmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ u(e+p) \end{vmatrix}, \quad G = \begin{vmatrix} \rho v \\ \rho u v \\ \rho v^2 + p \\ v(e+p) \end{vmatrix}.$$

Here u, v are the x, y velocity vector components, p is the pressure, ρ is the density and e is total energy of the fluid. We assume that the fluid is an ideal gas and the pressure is given by the equation-of-state.

$$p = (\gamma - 1)(e - \frac{\rho}{2}(u^2 + v^2)) \quad (2)$$

where γ is the ratio of specific heats and typically taken as 1.4 for air. It is assumed that an initial distribution of the fluid parameters is given at $t = 0$, and the boundary conditions defining a unique solution are specified for the computational domain.

The system of governing equations in Eq. (1) can be written as

$$\frac{\partial U}{\partial t} + \nabla \cdot Q = 0 \quad (3)$$

where Q represents the convective flux vector. By integrating Eq. (3) over space and using Gauss' theorem, the following expression is obtained

$$\frac{\partial}{\partial t}\int_{\Omega} U dA + \oint_{\partial\Omega} Q dl = 0 \quad (4)$$

where $dl = nd\mathcal{L}, n$ is the unit normal vector in the outward direction, and $d\mathcal{L}$ is a unit length on the boundary of the domain. The variable Ω is the domain of computation and $\partial\Omega$ is the circumference boundary of this domain.

Equation (4) can be discretized for each element (cell) in the domain

$$\frac{(U_i^{n+1} - U_i^n)}{\Delta t} A_i = \sum_{j=1}^{M} Q_j^n n_j \Delta L_j \quad (5)$$

where A_i is the area of the cell; Δt is the marching time step; U_i^{n+1} and U_i^n are the primitive variables at the center of the cell at time n and at the update $n = 1$ time step; Q_j is the value of the fluxes across the M boundaries on the circumference of the cell where n_j is the unit normal vector to the boundary edge j, and ΔL_j is the length of the boundary edge j. The fluxes Q_j^n are computed by applying the Second Order Godunov algorithm, and Eq. (5) is used to update the physical primitive variables U_i according to computed fluxes for each marching time step Δt. The marching time step is subjected to the CFL (Courant-Fredrichs-Lewy) constraint.

We seek a solution to the system of Eq. (1) in the computational domain, which is decomposed in part into triangles with arbitrary connectivity and in part into rectangles using a logically structured grid. We use the advantage of the unstructured grid[12-15] to describe the curved boundary of the computational domain and areas that need increased local resolution; this covers 10% of the total computational domain. The structured grid occupies the remaining 90% of the computational domain in our example. The numerical technique for solving Euler's equation on an unstructured grid is described in Refs. 9-10, and the technique for the structured grid is described in Ref. 11. These numerical techniques apply some of the ideas that were introduced in Refs. 17-18. The structured and unstructured codes apply the center-based formulation, i.e., the primitive variables are defined in the center of the cell, which makes the cell the integration volume, while the fluxes are computed across the edges of the cell. The basic algorithmic steps of the Second Order Godunov method can be defined as follows:

1. Find the value of the gradient at the baricenter of the cell for each gas dynamic parameter U_i;
2. Find the interpolated values of U at the edges of the cell using the gradient values;
3. Limit these interpolated values based on the monotonicity condition;
4. Subject the projected values to the characteristic's constraints;
5. Solve the Riemann problem by applying the projected values at the two sides of the edges;
6. Update the gas dynamic parameter U according to the conservation equations (1) applying to the fluxes computed and the current time step.

As was advocated in Ref. 9, we prefer the triangle center-based over the vertex-based version of the code. For the same unstructured grid, a triangle-based algorithm will result in smaller control volumes than a vertex-based. In addition, for the Second Order Godunov solver, implementation of the boundary conditions is more straightforward and accurate for the center-based algorithm than in the vertex-based. These two factors, along with the effects of grid connectivity, strongly affect the algorithm accuracy and performance, and are the main reasons for the superiority of the center-based version over the vertex version.

4. Results for Wing-PDE configuration

All of our previous studies considered axisymmetric configurations of the PDE devices. However, because PDE does not have rotating parts, it allows another degree of flexibility that enables us to configure the PDE devices in other than axisymmetric geometries. To illustrate this, we used the inner volume of a section of the wing as a detonation chamber for a PDE device. The schematic of the Wing-PDE geometry considered in this study is shown in Figure 4. We assume that the wing is located in a subsonic air flow stream with M = 0.8. The particular wing shape used is the Gastelow cusped supercritical airfoil.[12] Two significant modifications of the original Gastelow airfoil geometry, provision for an inlet

at the leading edge and an outlet nozzle at the trailing edge, allow its use as a PDE device.

In Figure 5, the cross section of the Wing-PDE geometry is shown in the computational domain that is decomposed into structured rectangular and unstructured triangular grids. For clarity, we show only every sixth point of the grid used in simulation. In our simulations we have used a structured grid with 255×131 nodes and an unstructured grid with 7229 nodes. The area covered by the unstructured grid is about 10% of the total area of the computational domain. It is obvious from Figure 5 that the unstructured grid is used in the regions of the computational domain having complex geometry, i.e., wing external and internal surfaces, inlet, and nozzle. The structured rectangular grid is used to cover the rest of the computational domain. As mentioned previously, this method of domain decomposition leads to the most efficient use of computer resources. Our results demonstrate that flow propagates through the interfaces between the triangular unstructured and rectangular structured sections seamlessly.

First, we have to examine the flow pattern for the steady state flow regime of the Wing-PDE device shown in Figure 5. This will also establish the reference values of the airdynamic drag and lift for this configuration. In Figure 6a, the results are shown in form of the pressure contours for the converged steady state solution for the Wing-PDE configuration in M=0.8 external flow stream at zero angle of attack. We can observe in Figure 6a a very complex internal/external flow pattern around Wing-PDE geometry. In addition to the shock wave near the trailing edge on the upper surface of the wing, we can observe two additional shock waves. One is created by the flow exiting from the inner volume of the wing through the nozzle at the trailing edge, and another is created at the flow inlet located under the leading edge. The air flow enters the inner volume of the wing through the inlet and creates a complex flow field with an average pressure of ≈ 1.0 atm. It is easy to improve the flow uniformity in the inner volume of the inlet geometry and geometry of the inner surfaces. However, these aspects of the Wing-PDE design will be considered in future studies; for the purposes of this paper, we examine only the main features of the Wing-PDE configuration. The air flow in the inner volume of the wing create considerable drag. By integrating the pressure over the inner and outer surface of the Wing-PDE configuration, we have calculated the basic air dynamic characteristics of this profile at M = 0.8 flow. The following values for the steady state flow:

Lift: $C_l = 0.18$; Drag: $C_d = -0.138$; Pitching Moment: $C_m = 0.034$.

We have assumed that at $t = 0$, the inner volume of the wing is filled with a detonable gas mixture. The detonation wave is initiated at the aft end of the inner volume of the wing by a planar front. The fuel chosen for these simulations was ethylene. The detonability limits of ethylene in air range from 4% to 12% concentrations by volume, and depend somewhat on temperature and pressure. We assume for the sake of simplicity that the fuel/air ratio is 6% by volume.

In Figure 6b, the pressure contours are shown at $t = 1.18 \times 10^{-4}$ sec. The propagation of the detonation front is planar. However, because of the curved inner walls of the wing, the detonation front reflects from the wall surfaces and the maximum pressure in the reflected waves reach 36.6 atm. However, this level of pressure is observed in a very small area of the detonation front where reflected or colliding transverse waves can cause a local maximum. The detonation wave velocity for this mixture is about 1800 m/sec.

In Figure 6c, the pressure contours are shown at the time $t = 5.24 \times 10^{-4}$ sec, shortly after the detonation front has reflected from the inner surface of the leading edge. Here the maximum pressure was dropped to 12.1 atm, the reflected shock is moving in the direction of the trailing edge, and the expansion of the detonation products through the inlet was created a semicircular shock wave that propagates in the opposite direction to the external flow stream. In Figure 6d at the time $t = 9.5 \times 10^{-4}$ sec, the reflected wave reaches the nozzle at the trailing edge, and expansion of the detonation products through this nozzle creates an additional shock wave that expands in the direction of the flow stream. When the original reflected shock has reached the converging area at the trailing edge, it will partially reflect and send a shock wave towards the inner surface of the leading edge. In Figure 6e, the pressure contours are shown at $t = 1.39 \times 10^{-3}$ sec. Here the shock waves created by the detonation products emitting from the inlet and nozzle of the Wing-PDE device collide, creating a complex flow pattern with two triple point shocks, a vortex at the trailing edge and a complex system of waves propagating through the inner volume of the wing. The maximum pressure observed in Figure 6e at the wave shock wave interaction is 3.2 atm. It is important to note that the numerical method simulates the flow evolution seamlessly through the structured/unstructured grid interfaces.

In Figure 6f, the simulation results are shown at $t = 5.7 \times 10^{-3}$ sec; this corresponds to the end of one cycle for the Wing-PDE configuration. Here we can observe that the flow pattern is very similar to the one in Figure 6a, except for some vortices propagating in the lower right part of the computational domain. The maximum pressure is reached at the leading edges and has the same values as shown in Figure 6a. The inner volume of the wing has a relatively uniform flow pattern

with an average pressure of 0.83 atm. At this time the gaseous mixture in the inner chamber of the wing will be initiated at the trailing edge and the second cycle will get started.

Examination of the details of the flow pattern resulting from a single detonation not only allows evaluation of the timing between the subsequent detonations but also provides important information for optimization of mixing, detonation products expansion, and other gasdynamic processes related to operation of the PDE cycle. Performance characteristics of the PDE device can be analyzed by integrating in time the forces exerted by pressure on the inner and outer surfaces of the Wing-PDE device. In Figure 7, results for such an integration of the force parallel to the ground as a function of time are shown. Calculation of this force, taking into account the drag and the thrust resulting from the detonation cycle, yields the net thrust force. Figure 7 gives this force for a linear meter of the wing in pounds. In this figure, we observe that the net thrust force is negative before the detonation is initiated, reaches the value of 4.6×10^5 Lb/M during the reflection of the main detonation front from the inner walls of the wing, and quickly decays to its negative initial values that correspond to the drag of the Wing-PDE configuration in M=0.8 ambient flow stream. The positive thrust force is produced by the detonation engine in a very short time interval; $\approx 3.0 \times 10^{-4}$ sec.

The time integral of the force shown in Figure 7 is thrust produced by the PDE device. Because of its intermittent operation, we need to assume the cycle frequency to be able to calculate the net thrust. In Figure 8, the results of thrust force integration are shown in the assumption of 200Hz detonation frequency of the Wing-PDE device. Our analysis above of a single cycle shows that this frequency of operation is feasible. In Figure 8, we observe that the maximum thrust of 5000 lb per linear meter of the wing is achieved in the first $\approx 4.0 \times 10^{-4}$ sec after the detonation wave impinges on the thrust wall. This period of time corresponds to the duration of the positive thrust force shown in Figure 7. After this, the thrust will erode because of drag force to the value of 4000 lb at the end of the cycle. The average thrust for the duration of the cycle is 4250 lb per linear meter of the wing.

One of the advantages of the Wing-PDE configuration is that it will generate lift. Our simulations show that the chosen configuration will produce significant lift even at zero angle of attack because of the flow of detonation products. In Figure 9, the net integrated lift is presented as a function of time in the same format as the net thrust shown in Figure 8. The integrated lift shown in Figure 9 is not a linear function of time, as will be the case for the steady state flow regime. Substantial lift is generated shortly after the detonation products start to expand into the surrounding flow stream. The average lift generated is about 2250 lb per meter of wing length: this is comparable to the net thrust of 4250 lb. Our estimates indicate that about half of this lift is generated by the detonation products and the other half by the free stream flow through the chamber.

5. Conclusions

We have presented a powerful numerical technique for analysis of nonsteady flow over a complex geometrical configuration in the computational domain decomposed on unstructured triangular and structured rectangular grids. Simulations of the Wing-PDE cycle have demonstrated flexibility and efficiency of this technique of domain decomposition. Numerical results show seamless propagations through structured/unstructured grid interfaces of the multiple shocks, contact discontinuities, vortices, rarefaction waves and other complex flow features.

Use of this powerful numerical technique allowed us to examine the operation cycle and propulsion characteristics of the Wing-PDE device. We demonstrated in this study that in principle, the Wing-PDE device can operate with the 200 Hz cycle frequency producing 4250 lb per linear meter of the wing of the net thrust. We examined the Wing-PDE configuration to illustrate the geometric flexibility of this engine. This is an additional advantage to efficiency,[3] scalability,[4] thrust control,[3] simplicity, and low cost of this device discussed in our previous publications.

References

1 Helman, D., Shreeve, R.P., and Eidelman, S., (1986), "Detonation Pulse Engine," AIAA-86-1683, 24^{th} Joint Propulsion Conference, Huntsville.

2 Eidelman, S., W. Grossmann, I. Lottati, (1989) "A Review of Propulsion Applications of the Pulsed Detonation Engine Concept," AIAA 89-2466, AIAA, July 10-12 (to be published in AIAA Journal of Propulsion), Nov - Dec issue.

3 Eidelman, S., W. Grossmann, and I. Lottati, (1990), "Computational Analysis of the Pulsed Detonation Engines and Applications," AIAA 90-0460, 28th Aerospace Sciences Meeting, Reno, NV, Jan 8-11.

4. Eidelman, S., W. Grossmann, and I. Lottati, "Air-Breathing Pulsed Detonation Engine Concept; A Numerical Study," AIAA/SAE/ASME/ASEE 26th Joint Propulsion Conference, Orlando, FL, July 16-18, 1990.

5. Eidelman, W., W. Grossmann, and I. Lottati, "A Propulsiohn Device Driven by Reflected Shock Waves," 18th International Symposium on Shock Waves, Sendai, Japan, 1991.

6 Hoffman, N., (1940), "Reaction Propulsion by Intermittent Detonative Combustion," Ministry of Sup-

ply, Volkenrode Translation.

7 Nicholls, J.A., Wilkinson, H.R., and Morrison, R.B., (1957), "Intermittent Detonation as a Thrust-Producing Mechanism," Jet Propulsion, 27, 534-541.

8 Nicholls, J.A., Gullen, R.E. and Ragland K.W., (1966), "Feasibility Studies of a Rotating Detonation Wave Rocket Motor," J. of Spacecrafts and Rockets, 3, 893-896.

9 Lottati, I., S. Eidelman, A. Drobot (1990a), "A Fast Unstructured Grid Second Order Godunov Solver (FUGGS)," Paper AIAA 90-0699, 28th Aerospace Sciences Meeting, Reno, NV, Jan 8-11.

10 Lottati, I., S. Eidelman, A. Drobot (1990b), "Solution of Euler's Equations on Adaptive Grids Using a FUGGS," to be published in Proceedings of Second International Conference on Free-Lagrange Methods, Jackson Hole, WY.

11 Eidelamn, S., Collela, P., and Shreeve R.P., (1984) "Application of the Godunov Method and Its Second Order Extension to Cascade Flow Modeling," AIAA Journal, v. 22, 10.

12. A. Jameson, T.J. Baker and N.P. Weatherill, "Calculation of Inviscid Transonic Flow Over a Complete Aircraft," AIAA 24th Aerospace Sciences Meeting, Reno, NV, AIAA Paper 86-0103, January 1986.

13. R. Lõhner, "Adaptive Remeshing for Transient Problems," Comp. Meth. Appl. Mech. Eng. 75, 195-214 (1989).

14. J. Peraire, M. Vahdati, K. Morgan and O.C. Zienkiewicz, "Adaptive Remeshing for Compressible Flow Computations," J. Comp. Phys. 72, 449-466, (1987).

15. D. Mavriplis, "Accurate Multigrid Solution of the Euler Equations on Unstructured and Adaptive Meshes," AIAA 88-3707 (1988).

16. I. Lottati and S. Eidelman, "Second Order Godunov Solver on Adaptive Unstructured Grids," Proceeding of the 4th International Symposium on Computational Fluid Dynamics, Davis, CA, September 1991.

17. B. van Leer, "Towards the Ultimate Conservative Difference Scheme, V.A. Second Order Sequel to Godunov's Method," J. Comp. Phys. 32, 101-136 (1979).

18. P. Collela and P. Woodward, "The Piecewise Parabolic Method (PPM) for Gasdynamic Simulations," J. Comp. Phys. 54, 174-201 (1984).

19 Dulikravich, D.S., (1982), Private communication.

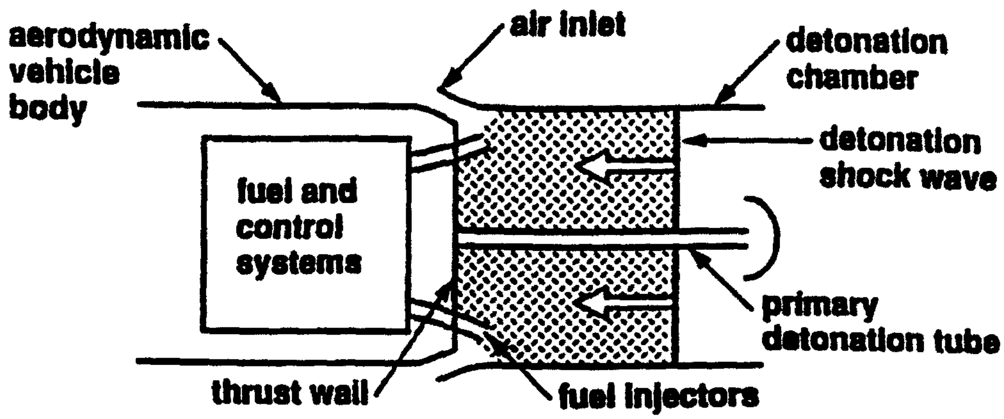


Figure 1. Schematic of the generic PDE showing detonation chamber, inlet, detonation wave, fuel injectors and position relative to an aerodynamic vehicle.

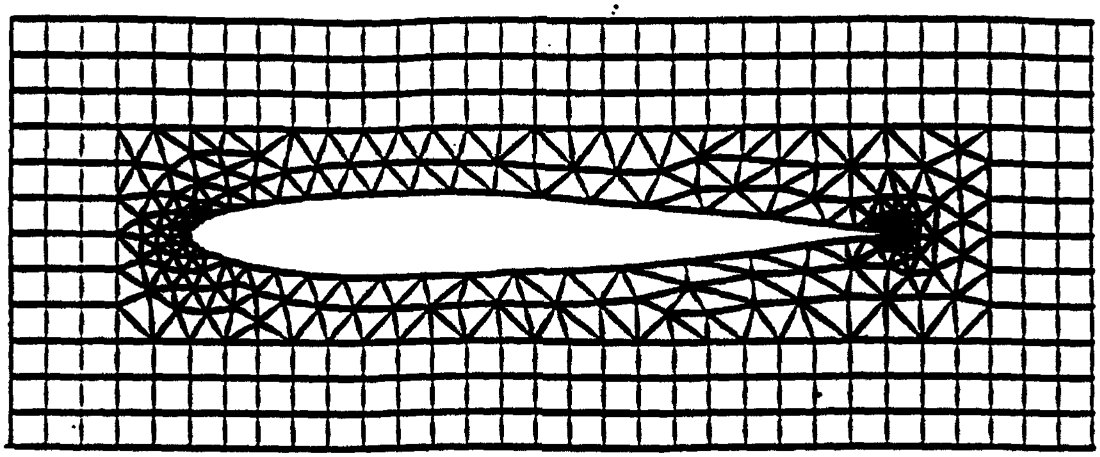


Figure 2. An example of hybrid structured/unstructured domain decomposition.

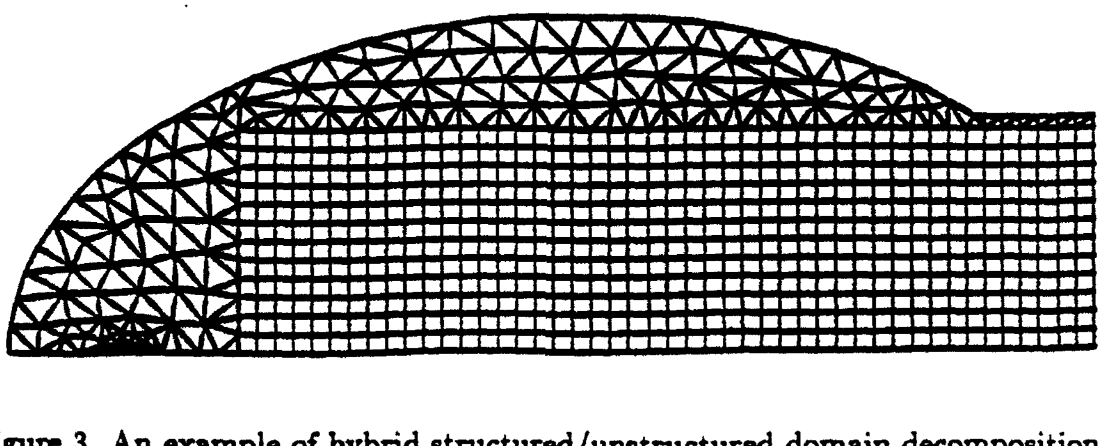


Figure 3. An example of hybrid structured/unstructured domain decomposition.

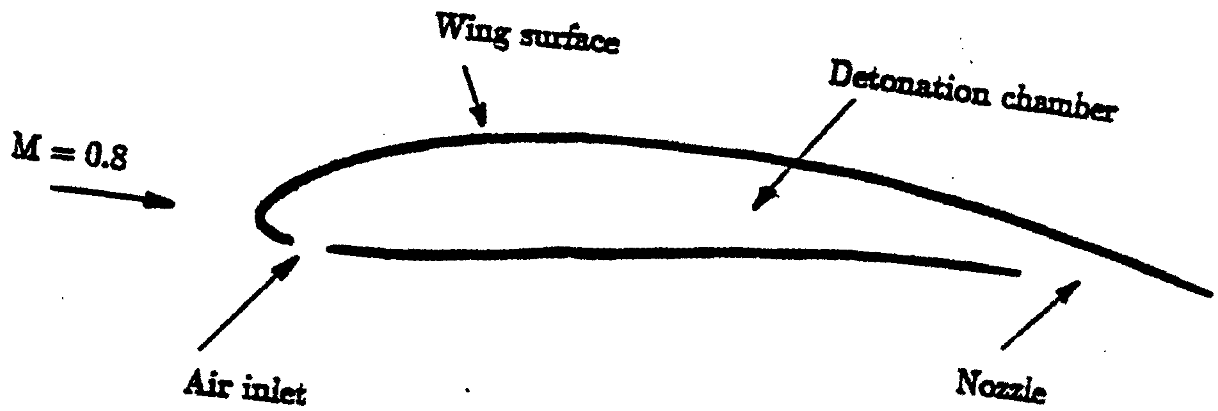


Figure 4. Schematics drawing of the wing-PDE configuration.

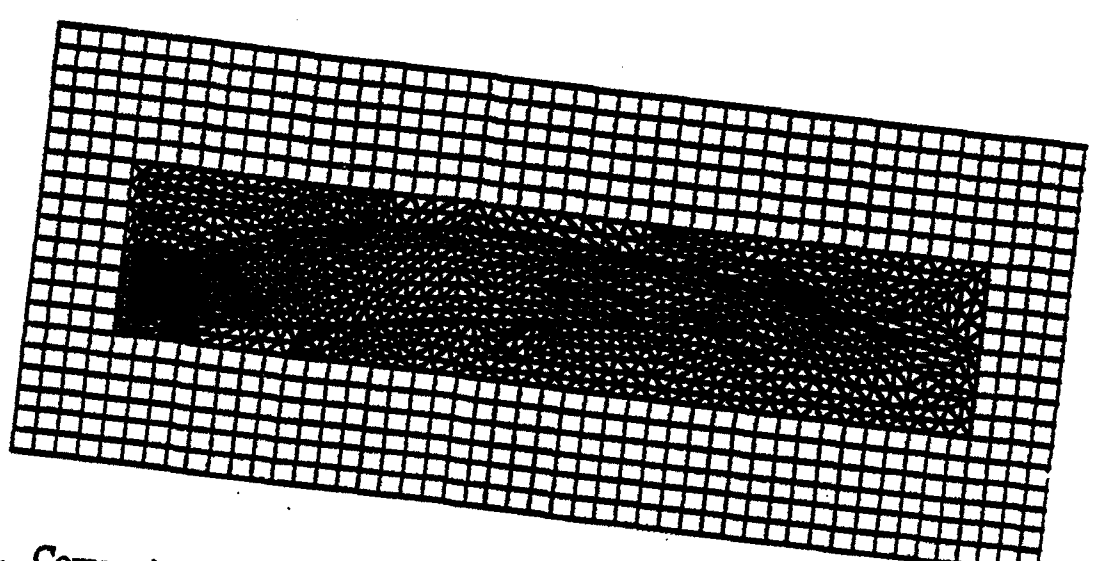
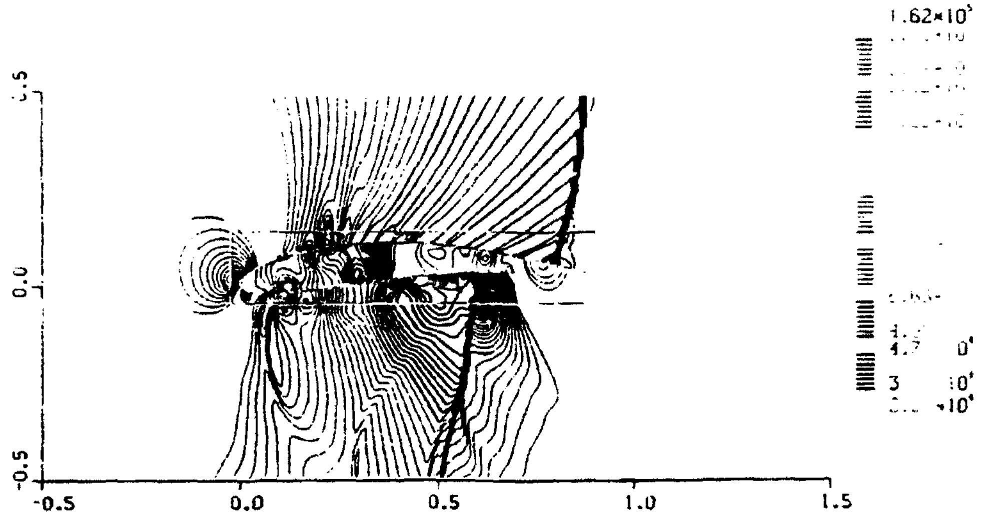
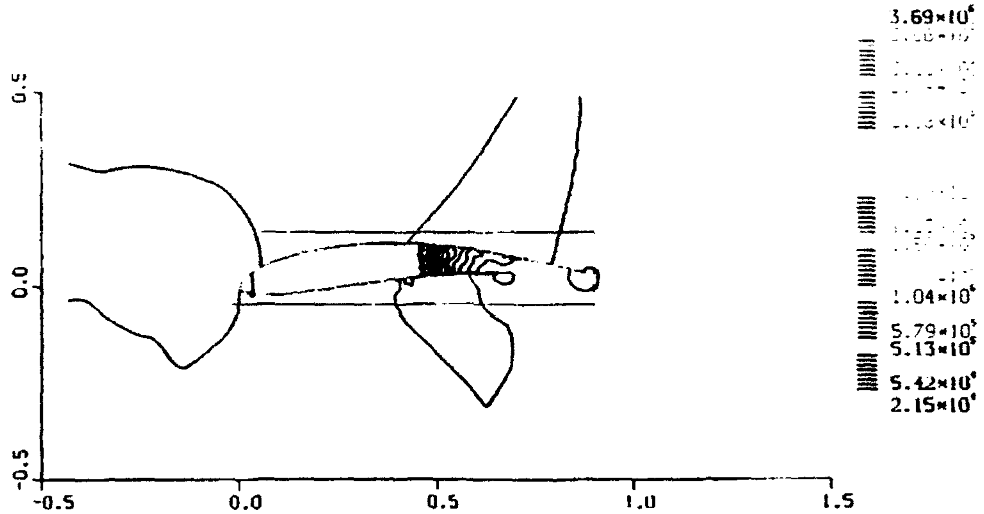


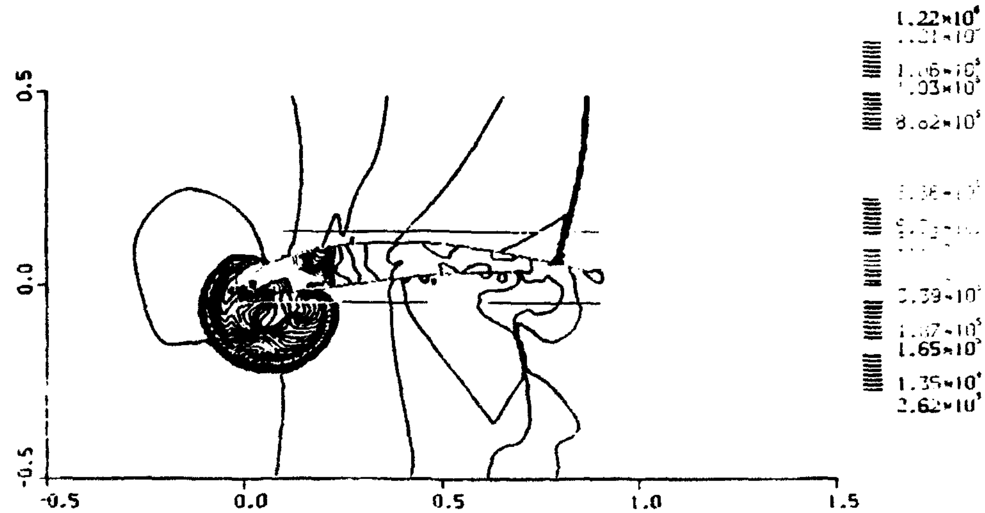
Figure 5. Composite structured/unstructured computational domain for the wing-PDE configuration.



a. Steady state

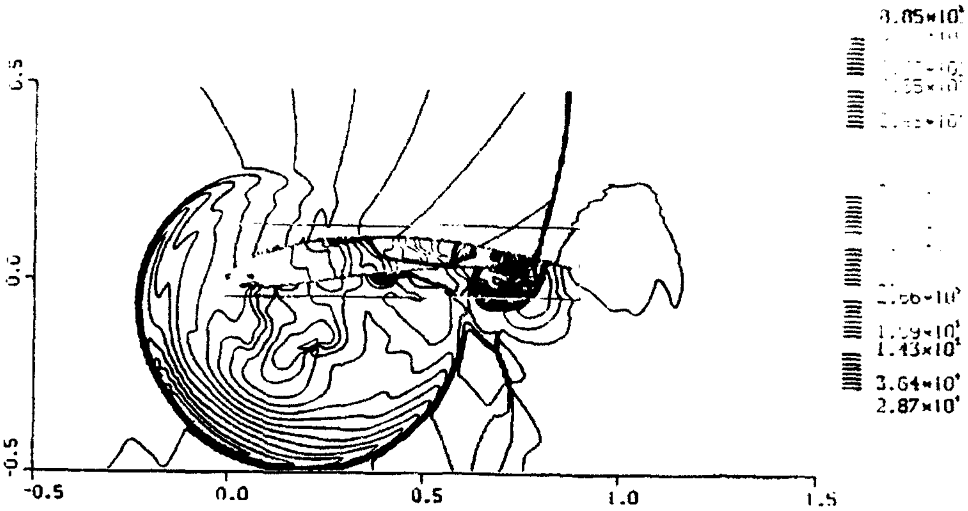


b. t = 1.18×10^{-4} sec

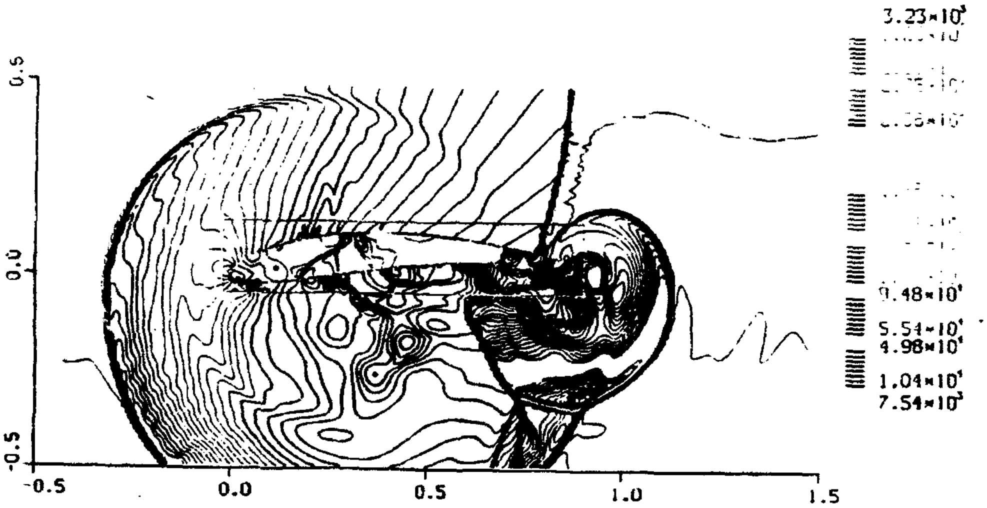


c. t = 5.24×10^{-4} sec

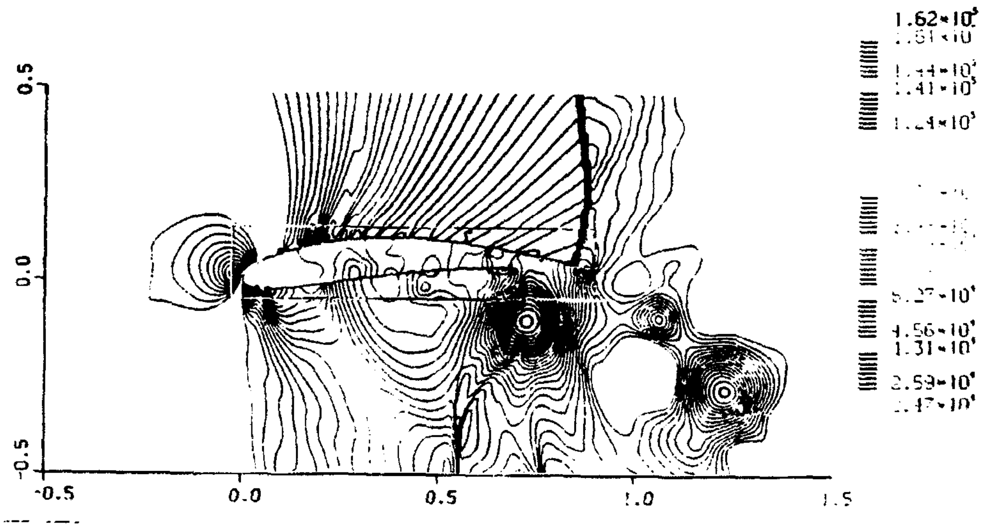
Figure 6. Pressure contours for the various time intervals of the wing-PDE cycle.



d. t = 9.50 × 10^{-4} sec



e. t = 1.39 × 10^{-3} sec



f. t = 5.77 × 10^{-3} sec

Figure 6. Pressure contours for the various time intervals of the wing-PDE cycle (continued).

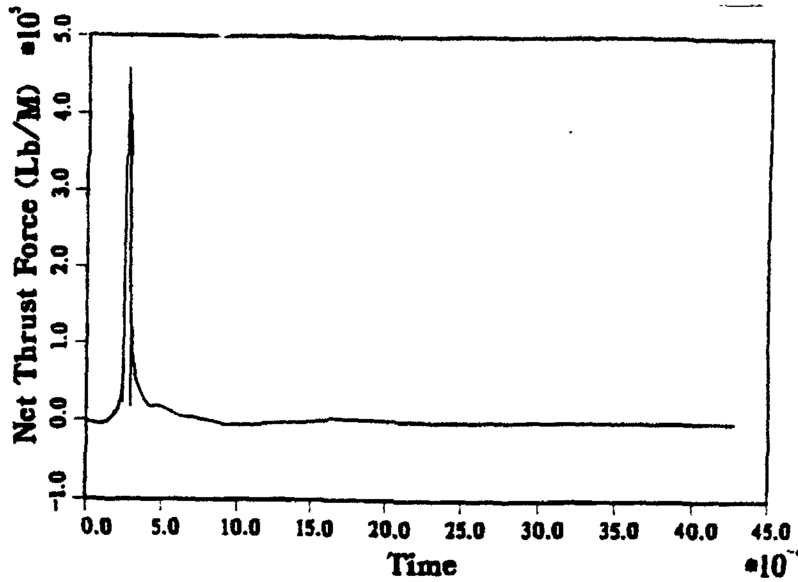


Figure 7. Thrust force as function of time for the wing-PDE device simulation.

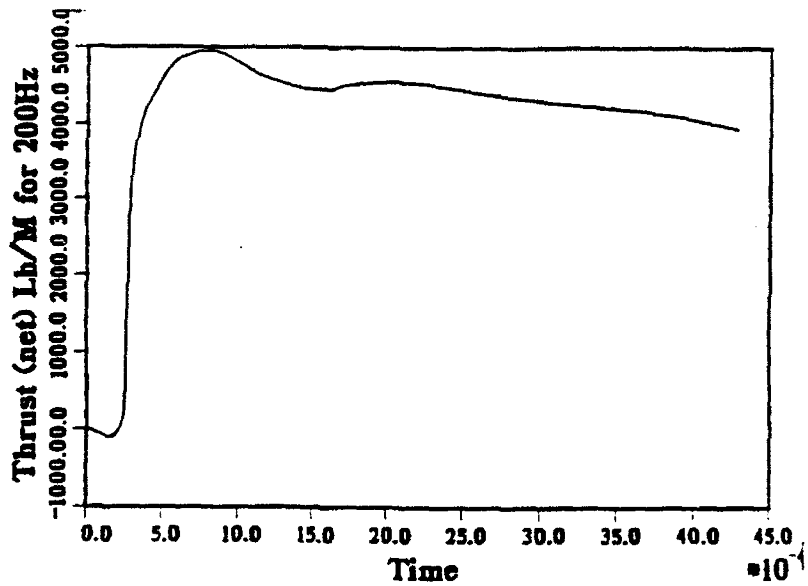


Figure 8. Net integrated thrust for the wing-PDE simulation.

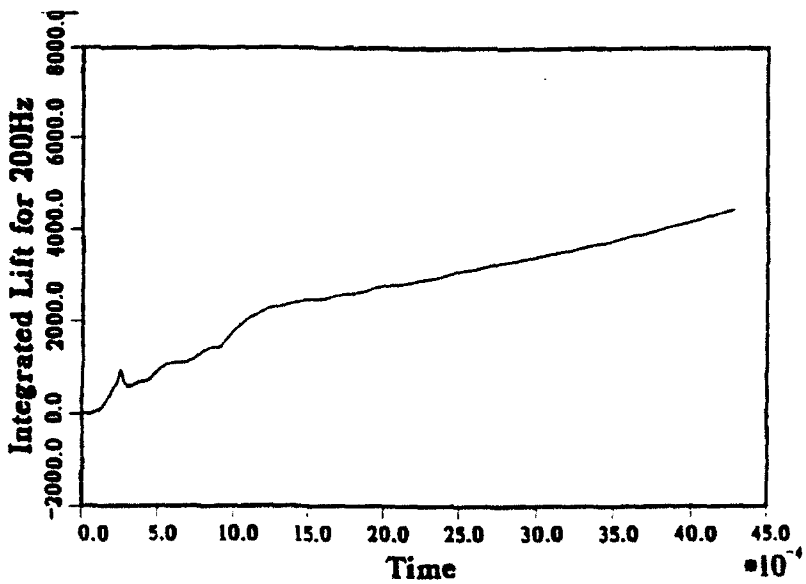


Figure 9. Integrated net lift for the wing PDE simulation.

A Second Order Godunov Scheme on Spatial Adapted Triangular Grid

Itzhak Lottati and Shmuel Eidelman

Science Applications International Corporation

ABSTRACT

Spatial adaptation procedure for the accurate and efficient solution of unsteady inviscid flow simulation is described. The adaptation procedures were developed and implemented applying a second order Godunov scheme. These procedures involve mesh enrichment/coarsening to either add/remove vertices in high/low gradient regions of the flow, respectively. The goal is to achieve solutions of high spatial accuracy at minimal computational cost. The paper describes a very effective error estimator to detect high/low activity regions of the flow to be enriched or coarsened, respectively. The error estimator is based on total energy and density fluxes into the cell combined with gradient of density. Included in the paper is a detailed description of the direct dynamic refinement method that is used for adaptation. A detailed simulation of a reflection and diffraction of multiple shock waves flowing over a diamond shape wedge is presented and compared with experimental results. The simulated results are shown to be in excellent agreement with the experiment primarily in that all the complicated features of the physics are accurately accounted for and the shock waves, slip lines, vortices are sharply captured.

INTRODUCTION

Considerable progress has been made over the past decade in developing methods for spatial adaptation of the computational meshes based on the numerical solution of the simulated physics. These methods are being developed to produce higher spatial accuracy in such simulation more efficiently. The goal of mesh adaptation is to enrich meshes locally, based on the numerical solution, in order to capture physical features of importance; in contrast to globally fine meshes, this process will minimize computer run times and memory costs. The methods of mesh adaptation can be categorized into three general classes: 1) mesh regeneration, 2) mesh movement, and 3) mesh enrichment.

The idea of mesh regeneration is systematically to identify high/low activity region in the flow and accordingly remesh those regions applying mesh generation code. This is done by assigning criteria for spatial accuracy and number of vertices. This procedure requires a mapping of the "old" flow solution into the "new" generated meshes by using one of the interpolated schemes. For the second method, mesh movement, the number of points in the computational domain remains fixed. The adaptation procedure moves vertices from low activity regions to high gradient regions to achieve a high concentration of vertices to resolve high activity regions. The movement of the points is dictated by forcing functions in the Poisson - equation in the grid generator code. The final method of spatial adaptation is mesh

enrichment. In this method, vertices are added or removed according to the spatial resolution of the physical features in the flow. The advantages of mesh enrichment over regeneration and movement are its higher degree of flexibility in being able to add points where they are needed and to remove points where they are not needed. In our mesh enrichment method, we add points ahead of the shock wave, thus preventing the need of interpolation in the high gradient region for achieving higher accuracy of the results. Adding and removing points are done in monotone/very low activity regions to prevent numerical dissipation.

Lohner(1) has developed procedures to enrich the mesh for transient flow problems locally by subdividing elements in the grid according to specific spatial resolution criteria. The method, referred to as H-refinement, keeps a history of the initial grid (mother grid) and the subdivision of each level (daughter grids). The H-refinement relies heavily on the initial grid as it is subdivided for enrichment and recovered in the coarsening stage. A similar adaptive strategy to Lohner is adopted by Rausch(2) et al., but applies a different error estimator and upwind type algorithm for a solver.

In our paper, we describe a Godunov scheme to solve Euler equations on an unstructured adaptive triangle mesh. We discuss the methodology of a cell centered Second Order Godunov scheme applied to a triangular mesh, and the method of Direct Dynamic Refinement that is used for adaptation of the unstructured triangular grid. Simulation and experimental results

are compared for a test case applying the adaptive unstructured grid to a complicated pattern of planar shock wave flow diffraction over a half diamond shape wedge.

SECOND ORDER GODUNOV ALGORITHM ON UNSTRUCTURED GRID

This section describes the implementation of the Second Order Godunov algorithm on a triangular unstructured grid. The algorithm is explicit and is cell-center based.

We consider a system of two-dimensional Euler equations written in conservation law form as:

$$\frac{\partial \bar{U}}{\partial t} + \frac{\partial \bar{F}}{\partial x} + \frac{\partial \bar{G}}{\partial y} = 0 \qquad (1)$$

where

$$U = \begin{Bmatrix} \rho \\ \rho u \\ \rho v \\ e \end{Bmatrix}, F = \begin{Bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ u(e+p) \end{Bmatrix}, G = \begin{Bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ v(e+p) \end{Bmatrix}.$$

Here u, v are the x, y velocity vector components, p is the pressure, ρ is the density and e is total energy of the fluid. We assume that the fluid is an ideal gas. The total energy of gas is given by the following equation:

$$e = \frac{p}{\gamma - 1} + \frac{\rho}{2}(u^2 + v^2) \qquad (2)$$

where γ is the ratio of specific heats. It is assumed that an initial distribution of the fluid parameters is given at $t = 0$, and the boundary conditions defining a unique solution are specified for the computational domain.

The system of governing equation (1) can be written in the following form:

$$\frac{\partial U}{\partial t} + \bar{\nabla} \cdot \bar{Q} = 0 \tag{3}$$

where $\bar{Q}$ represents the convective flux vector. By integrating Eq. (3) over space and using Gauss' theorem, the following expression is obtained

$$\frac{\partial}{\partial t} \int_{\Omega} U dA + \oint_{\partial\Omega} \bar{Q} \cdot d\bar{l} = 0 \tag{4}$$

where $d\bar{l} = \bar{n} d\mathcal{L}, \bar{n}$ is the unit normal vector in the outward direction, and $d\mathcal{L}$ is a unit length on the boundary of the domain. The variable Ω is the domain of computation and $\partial\Omega$ is the circumference boundary of this domain.

Equation (4) can be discretized for each element (cell) of the domain

$$\frac{(U_i^{n+1} - U_i^n)}{\Delta t} A_i = \sum_{j=1}^{3} \bar{Q}_j^{n+\frac{1}{2}} \bar{n}_j \Delta L_j \tag{5}$$

where A_i is the area of the cell; Δt is the marching time step; U_i^{n+1} and U_i^n are the primitive variables at the center of the cell at time n and at the update $n+1$ time step; $\bar{Q}_j$ is the value of the fluxes across the three boundaries edges on the circumference of the cell where $\bar{n}_j$ is the unit normal

vector to the boundary edge j, and ΔL_j is the length of the boundary edge j. Equation (5) is used to update the physical primitive variables U_i according to computed fluxes for each time step Δt. The time step is subjected to the CFL (Courant-Fredrichs-Lewy) constraint.

To obtain a second order spacial accuracy, the gradient of each primitive variable is computed in the baricenter of the cell. This gradient is used to define the projected values of primitive variables at the two sides of the cell's edge, as is shown in Figure 1. The gradient is approximate by a path integral

$$\int_{\Omega} \bar{\nabla} U_i^{cell} dA = \oint_{\partial\Omega} U_j^{edge} d\bar{l} \,. \tag{6}$$

The notation is similar to the one used for Eq. (5) except the domain Ω is a single cell and U_i^{cell} and U_j^{edge} are values at the baricenter and on the edge respectively. The gradient is estimated as

$$\bar{\nabla} U_i^{cell} = \frac{1}{A} \sum_{j=1}^{3} \tilde{U}_j^{edge} \bar{n}_j \Delta L_j \tag{7}$$

where $\tilde{U}_j^{edge}$ is an average value representing the primitive variable value for edge j.

The gradients that are computed at each baricenter are used to project values for the two sides of each edge by piecewise linear interpolation. The interpolated values are subjected to monotonicity constraints.[3] The monotonicity constraint assures that the interpolated values are not creating new

extrema.

The monotonicity limiter algorithm can be written in the following form:

$$U^{edge}_{projected} = U^{cell}_i + \phi \bar{\nabla} U_i \cdot \Delta \bar{r} \tag{8}$$

where $\Delta \bar{r}$ is the vector from the baricenter to the point of intersection of the edge with the line connecting the baricenters of the cells over the two sides of this edge. ϕ is the limiter coefficient that limits the gradient $\bar{\nabla} U_i$.

First, we compute the maximum and minimum values of the primitive variable in the i's cell and its three neighboring cells that share common edges (see Fig. 1):

$$\left.\begin{aligned} U^{\max}_{cell} &= Max\left(U^{cell}_k\right) \\ U^{\min}_{cell} &= Min\left(U^{cell}_k\right) \end{aligned}\right\} k = i, 1, 2, 3\ . \tag{9}$$

The limiter can be defined as:

$$\phi = Min\left\{1, \phi^{lr}_k\right\} \ \ k = 1, 2, 3 \tag{10}$$

where superscript lr stands for left and right of the three edges (6 combinations in total). ϕ^{lr}_k is defined by:

$$\phi^{lr}_k = \frac{\left[1 + Sgn\left(\Delta U^{lr}_k\right)\right] \Delta U^{\max}_{cell} + \left[1 - Sgn\left(\Delta U^{lr}_k\right)\right] \Delta U^{\min}_{cell}}{2(\Delta U^{lr}_k)} \quad k = 1, 2, 3 \tag{11}$$

where $\Delta U_k^{lr} = \bar{\nabla} U_i^{lr} \cdot \Delta \bar{r}_k$. and

$$\left.\begin{aligned} \Delta U_{cell}^{max} &= U_{cell}^{max} - U_i^{cell} \\ \Delta U_{cell}^{min} &= U_{cell}^{min} - U_i^{cell} \end{aligned}\right\} . \tag{12}$$

To obtain a second order of accuracy in time and space, we subject the projected values of the left and right side of the cell edge to characteristic constraints following Ref. 4. The one dimensional characteristic predictor is applied to the projected values at half time step $t^n + \frac{\Delta t}{2}$. The characteristic predictor is formulated in the local system of coordinates for the one dimensional Euler equation. We illustrate the implementation of the characteristic predictor in the direction of the unit vector $\bar{n}_c$. The Euler equations for this direction can be written in the following form:

$$W_t + A(W) W_{nc} = 0 \tag{13}$$

where

$$W = \begin{Bmatrix} \tau \\ u \\ p \end{Bmatrix}; \ A(W) = \begin{pmatrix} u & -\tau & 0 \\ 0 & u & \tau \\ 0 & \rho c^2 & u \end{pmatrix} \tag{14}$$

where $\tau = \rho^{-1}$, ρ denotes density while u, p are the velocity and pressure. The matrix $A(W)$ has three eigenvectors $(l^{\#}, r^{\#})$ (l for left and r for right where # denote +,o,−) associated with the eigenvalues $\lambda^+ = u + c$, $\lambda^o = u$, $\lambda^- = u - c$.

An approximation of projected value to an edge accurate to second order in space and time can be written as:

$$W_{i+\Delta r}^{n+1/2} \approx W_i^n + \frac{\Delta t}{2}\frac{\partial W}{\partial t} + \Delta r \frac{\partial W}{\partial r_{nc}}$$
$$\approx W_i^n + \left[\Delta r - \frac{\Delta t}{2}A(W_i)\right]\frac{\partial W}{\partial r_{nc}} \tag{15}$$

An approximation to $W_{i+\Delta r}^{n+1/2}$ can be written as:

$$W_{i+\Delta r}^{n+1/2} = W_i + (\Delta \bar{r}_i - \frac{\Delta t}{2}(M_x M_n)\cdot \bar{n}_c)\bar{\nabla} W_i \tag{16}$$

where

$$(M_x M_n) = \begin{cases} Max(\lambda_i^+, o) & \text{for cell left to the edge} \\ Min(\lambda_i^-, o) & \text{for cell right to the edge .} \end{cases} \tag{17}$$

The gradients applied in the process of computing the projected values at $t^n + (\Delta t/2)$ are subjected to the monotonicity limiter.

Following the characteristic predictor described above, the full Riemann problem is solved at the edge. The solution of the Riemann problem defines the flux $\bar{Q}^{n+\frac{1}{2}}$ through the edge. The fluxes through the edges of triangles are then integrated (Eq. 5), thus giving an updated value of the variables at t^{n+1}. One of the advantages of the described algorithm is that calculation of the fluxes is done over the largest loop in the system (loop over edges) and can be carried out in the vectorized or parallelized loop. This fact leads to an efficient algorithm.

The algorithm presented is a modification of the algorithm of Ref. 5 which was derived for structured mesh. This algorithm has been applied to simulate a wide range of flow problems and has been found very accurate in predicting the features of the physics. The performance of the algorithm is well documented in Refs. 6-8. The next section, the spatial adaptive procedure, is described in detail. These descriptions include explanations of the error estimator for flow feature detection and the Direct Dynamic Refinement Method used to enrich and coarsen the mesh.

DIRECT DYNAMIC REFINEMENT METHOD FOR ADAPTATION ON AN UNSTRUCTURED TRIANGULAR GRID

The Direct Dynamic Refinement method (DDR) is a new method for adapting unstructured triangular grids during the computational process. As stated, an unstructured grid is very suitable for implementing boundary conditions on complex geometrical shapes as well as the adaptation of the grid, if necessary. The adaptation of the unstructured triangular grid leads to efficient usage of memory resources. The adaptive grid enables the user to capture moving shocks and high gradient flow features with high resolution. The available memory resources can be very efficiently distributed in the computational domain to accommodate the resolution needed to capture features of the physical property of the solution as they are evolved. Dynamic refinement controls the resolution priorities. These priorities can be set according to the physical features that the user wishes to emphasize

in the simulation. The user has control over the accuracy of the physical features resolved in the simulation, without being restricted to the initial grid. The alternative to Direct Dynamic Refinement (DDR) is the hierarchical dynamic refinement (H-refinement) that keeps a history of the initial grid (mother grid) and the subdivision of each level (daughter grids). In the H-refinement method, it is necessary to keep overhead information on the level of each triangle subdivision, and double indirect indexing is needed to keep track of the H-refinement process. As mentioned, the H-refinement relies heavily on the initial grid as it subdivides this grid and returns to it after the passage of the shock.

To minimize the dissipation caused by the interpolation and extrapolation in the refinement and coarsening of the grid, the addition and deletion of point is done in the region where the flow features are smooth. Thus for capturing the shock, the refinement should be applied in the region ahead of the shock. The coarsening of the grid is done in the flow regions where the gradients of the flow parameters are small.

In the present version of AUGUST (Adaptive Unstructured Godunov Upwind Second order Triangular), we implemented an algorithm with multiple criteria for capturing a variety of features that might exist in the physics of the problem to be solved. To identify the location of a moving shock, we use the flux of total energy into triangles. The fluxes entering and leaving triangles are the most accurate physical variables computed by the Godunov

algorithm for solving Euler's equations, and are used to update the physical variables for each time step in each triangle. Supplementary to the : ux of energy as an error indicator, we use the flux of total density into t · ngles and the density gradient. The error indicator is the only sensor tha· is solely responsible for identifying the area to be refined or coarsened in tne computational domain. As such, the error indicator should be sensitive enough to detect physical features that are of interest to the user, such as shock waves, rarefaction waves, slip lines and vortices. The error indicators that are implemented in the code are able to sense very weak slip lines in the presence of strong shock waves. The ability of the error indicators to identify weak physical features in the presence of strong ones, without picking up numerical noises, is essential to the simulation of adaptive grids. As stated, the quality of the results is as good as the error indicators applied. If the error indicators fail to identify the physical feature, this feature probably will be overlooked in the simulated results. It should be noted that the process of applying error indicators for identifying the areas to be adaptively refined or coarsened is an expensive loop that has to check the whole triangles table in the simulation. Thus, the error indicators are applied each 9 to 15 time steps. This process is preceded by application of an algorithm that refines a buffer zone ahead of the features and coarsens the grid after it was moved away. The buffer zone ahead of the feature is identified by using a search pattern of finding the neighbors of the flagged triangles sorted by the error indicators.

We are not applying any physical parameters to identify the zones "ahead."

The refinement algorithm follows several basic steps. The process of adding points to refine the grid locally is done by either adding a new vertex in the baricenter of the triangle or adding a new vertex in the middle of the edge. Adding a new vertex in the baricenter of a triangle is very efficient in the sense that the refinement affects this individual triangle only. We apply this process exclusively for refinement. As a supplement, especially on the boundary, we apply the method of adding a new vertex on an edge. As a complement to adding new vertices, we apply the reconnection/swapping algorithm that flips the diagonal (common edge) of two adjacent triangles to improve the quality of the triangles constructed. Figure 2 displays a chain of those basic steps to illustrate the refinement process. Figure 2a shows the original grid. Figure 2b illustrates a one step scheme refinement in which a new vertex is introduced into a triangular cell forming three cells (two new ones). On the boundary edges, a new vertex is introduced in the middle of those edges to form two cells (one new one). This refinement is followed by reconnection that modifies the grid as demonstrated in Fig. 2c. The process of refinement and reconnection can be continued until the necessary grid resolution is achieved. As an example, another loop of refinement is illustrated in Figs. 2d and 2e. This direct approach to grid refinement provides extreme flexibility in resolving local flow features.

A similar direct approach is applied to grid coarsening. The basic step

in this process is deleting the cells and edges associated with a vertex to be removed, as shown in Fig. 3b. During the second step, this void in he grid is filled with new larger triangles (Fig. 3c) without introducing ne ertices. The last step is local reconnection and relaxation as shown in F . 3d. The relaxation procedure is a simple relocation of the vertex moved the center of the polygon surrounding this vertex (only if the polygon is a convex).

The algorithm of direct dynamic refinement proved to be very efficient in refining and coarsening the grid adaptively. The refinement and coarsening followed a short inquiry on the quality and shape of the triangle flagged and its close neighbors. Since we do not keep any history or tree for each triangle, the DDRM algorithm has much less checking to do as compared to the H-refinement algorithm. The vectorization and parallelization of the solver is straightforward.

NUMERICAL RESULTS FOR THE TWO DIMENSIONAL TEST PROBLEM

We have tested the Second Order Godunov algorithm in a variety of flow simulations ranging from the low subsonic to the high hypersonic Mach [6–8] regime. The AUGUST code proved to be very robust and accurate. The results obtained are comparable to or better than those obtained applying leading flow solvers in all of the regimes tested.

To validate our DDRM implemented in the AUGUST code, we simulated the problem of interaction of a Mach 2.85 planar shock wave, propagating

are accurately accounted for and the shock waves, slip lines, vortices are sharply captured.

in a channel with a 45° symmetrical double ramp. Figure 4 shows the experimental interferogram of the problem to be simulated (reproduced). The example that we chose to simulate is most appropriate to test the performance of an adaptive algorithm. The experimental results show a complex flow pattern containing a mix of strong discontinuities, as shock waves, and very weak features such as slip lines, vortices, and rarefaction waves. The error estimator must recognize and flag all these features for refinement. The error estimator should be sensitive enough to identify very weak slip lines without picking up numerical noises present in the simulation. We have simulated the shock wave reflection and diffraction over a 45° corner at the conditions that correspond to the experimental result shown in Fig. 4. Here we present results for several shapes of the flow evolution. The flow in the channel is from left to right. Figure 5 displays density contour plots after the shock passed the apex of the double wedge obstacle. In Fig. 5a, the density contours are overlayed on the grid used at this stage of the evolving flow. For clarity, only the density contours are displayed in Fig. 5b. The grid displayed in Fig. 5a shows how well the adaptation technique follows the high activity region in the flow. The grid is adapting to regions with high pressure gradients and high density gradient. In Fig. 5a, one can observe high quality grid produced by the DDR method. The shock has a relatively thin buffer zone ahead of its front, allowing us to avoid the interpolations related to grid adaptation of the flow variables in the area of high gradient.

The flow features are resolved accurately, and the contact discontinuity and triple point are clearly defined.

Figure 6 shows the density contours at a later time in the same format as in Fig. 5. This figure demonstrates the ability of the DDRM to identify and follow flow features in the computational domain. In this figure we can observe a complicated flow pattern developing as a result of interaction of the rarefraction wave with the complex pattern of shock waves. A recompression shock and a strong vortex that are developed in this time frame are well resolved. We can also observe a slip line originating at the triple point. The adaptation algorithm, as in the previous time frame, follows both shock waves and contact discontinuities.

Figure 7 displays the density contours at the stage comparable to that shown in Fig. 4 for the experimental results. The computed results as displayed in Fig. 7b show a flow pattern similar to the experiment. The slip line and the formation of vertices along it are clearly depicted. The shock and reflected shock as well as the recompression shock are very sharply defined with very low numerical noise. The vortex developed after the compression shock is distinctly displayed. A new reflected shock can be seen developing at the channel wall behind the double wedge.

The results shown in Figs. 5-7 display the ability of the algorithm to simulate a complex transient flow problem on dynamically adapting grid. The error estimates used in our algorithm allow detection of strong and

weak shock waves, conducted discontinuities, vortices or other fronts that need enhanced resolution.

CONCLUSION

The Direct Dynamic Refinement (DDR) method was developed and tested for a challenging problem of reflection and diffraction of a strong shock over a double ramp. For this test problem we have demonstrated that a set of error indicators developed for the DDR allow capturing strong and weak features of the complex wave structure developing in this test case.

The above described algorithms were implemented in the AUGUST code. The AUGUST code was used for a range of subsonic, transonic, and supersonic transient and steady problems. For all these conditions the AUGUST code produced robust results with the error indicators proving to be applicable for all these diverse flow regimes.

ACKNOWLEDGMENT

The work reported here was partially supported by DARPA and AFOSR under Contract #F49620-89-C-0087. The authors would like to thank Col. James Crowley and Dr. A. Nachman for their interest in this project.

REFERENCES

1) R. Lohner, "An Adaptive Finite Element Scheme for Transient Problems in CFD," Comp. Meth. Appl. Mech. Eng., Vol. 61, 323-338, 1987.

2. R.D. Rausch, J.T. Batina, H.T.Y. Yang "Spatial Adaptation of Unstructured Meshes for Unsteady Aerodynamic Flow Computation," AIAA Journal, Vol 30, No. 5, May 1992, pp. 1243-1251.

3) B. Van Leer, "Toward the Ultimate Conservative Difference Scheme, V. A Second Order Sequel to Godunov Method," J. Comp. Phys., 32, 101-136, 1979.

4) P. Collela and H.M. Glaz, "Efficient Solution Algorithm for the Riemann Problem for Real Gases," J. Comp. Phys. 59, 264-289, 1985.

5) S. Eidelman, P. Collela, and R.P. Shreeve, "Application of the Godunov Method and its Second Order Extension to Cascade Flow Modeling," AIAA Journal 22, 10, 1984.

6) I. Lottati, S. Eidelman, and A.T. Drobot, "A Fast Unstructured Grid Second Order Godunov Solver (FUGGS)," AIAA 90-0699, 27th Aerospace Sciences Meeting, Reno, Nevada, 1989.

7) I. Lottati, S. Eidelman, and A.T. Drobot, "Solution of Euler's Equations on Adaptive Grids Using a Fast Unstructured Grid Second Order Godunov Solver," Proceedings of the Free Lagrange Conference, Jackson Lake, WY, June 1990.

8) I. Lottati and S. Eidelman, "Second Order Godunov Solver on Adaptive Unstructured Triangular Grid," Proceedings of 4th International Symposium on Computational Fluid Dynamics, Davis, CA, September 1991.

9. D.L. Zhang, I.I. Glass, "An Interferometric Investigation of the Diffraction of Planar Shock Waves Over a Half-Diamond Cylinder in Air," NTIAS Report No. 322, March 1988.

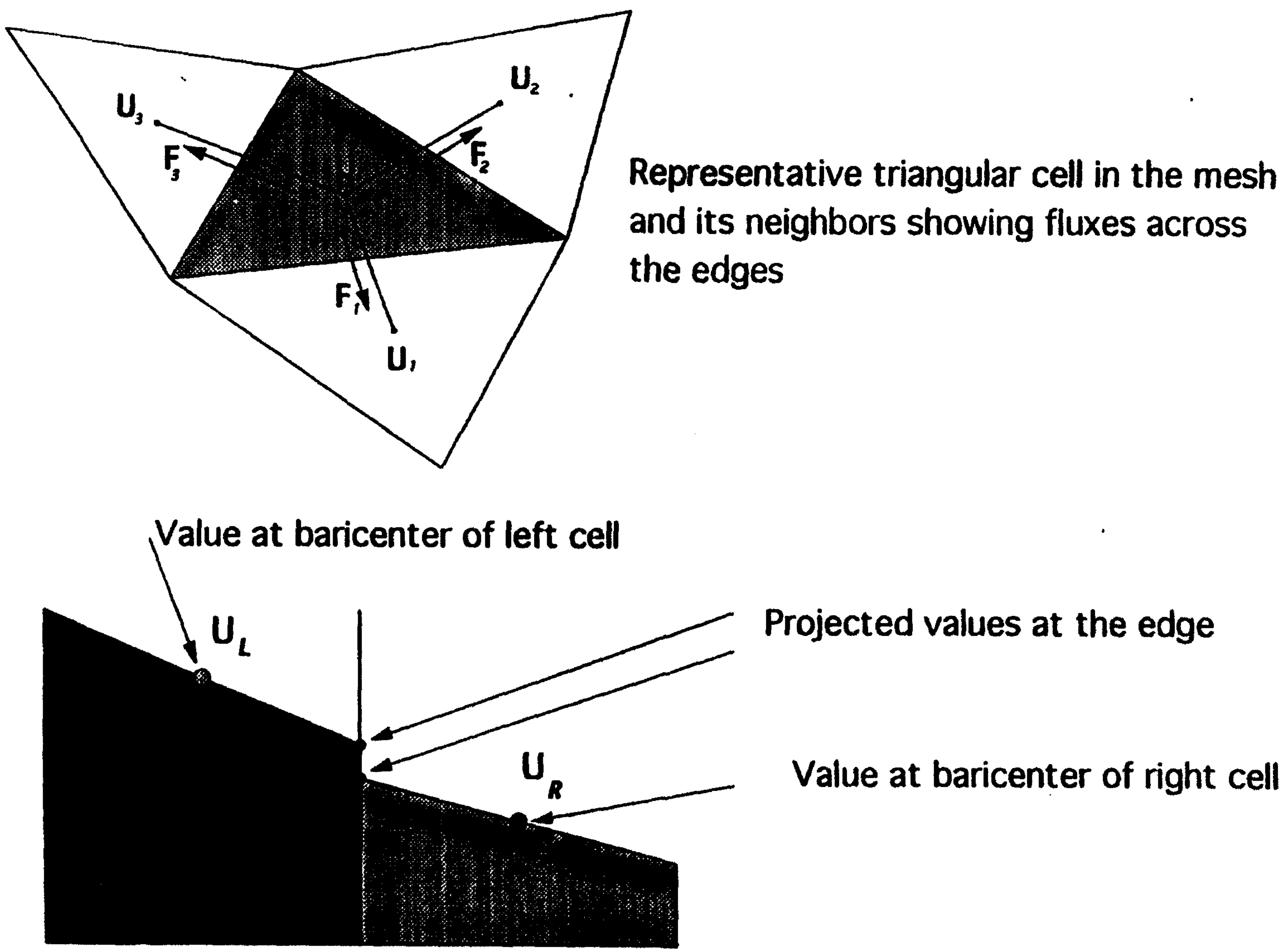
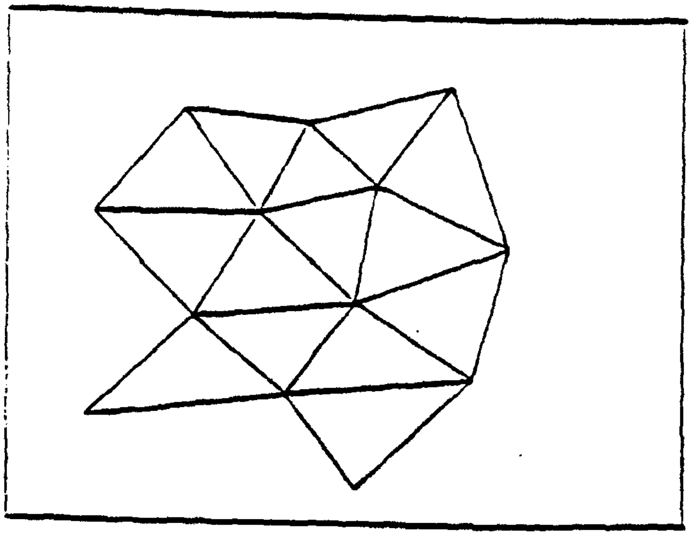
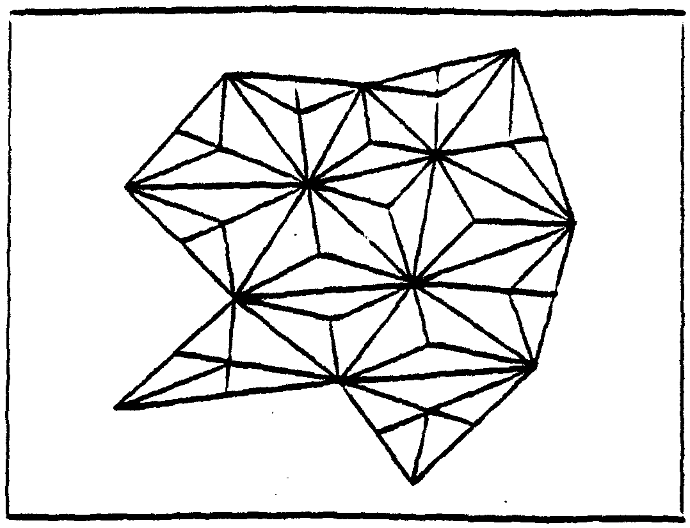


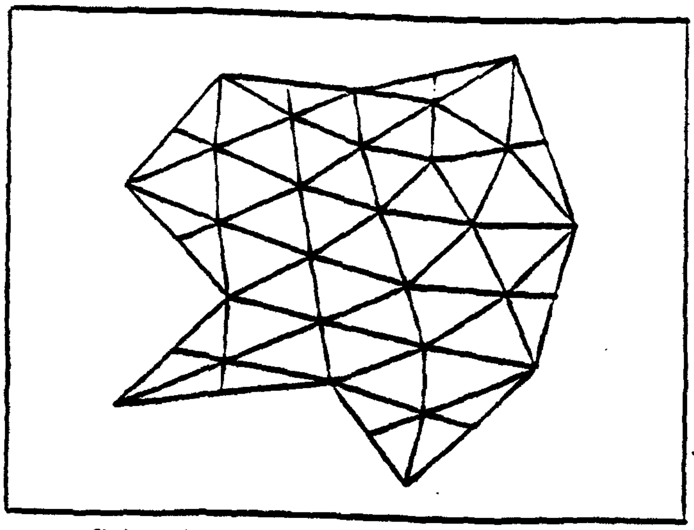
Figure 1. Representative triangular cell in the mesh showing fluxes and projected values.



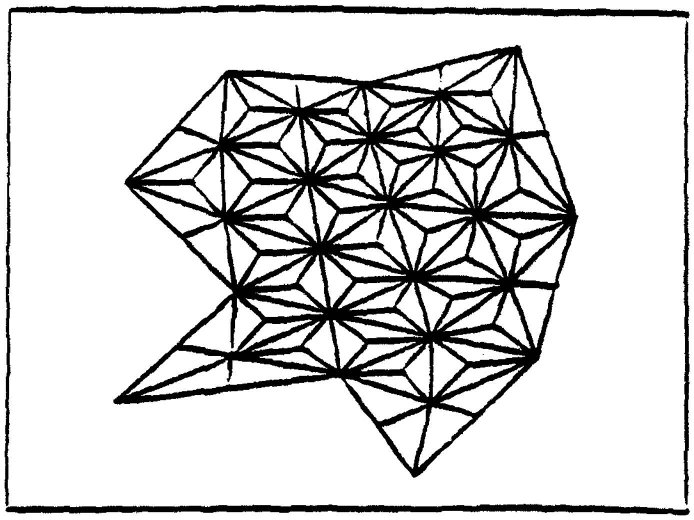
a. Original grid.



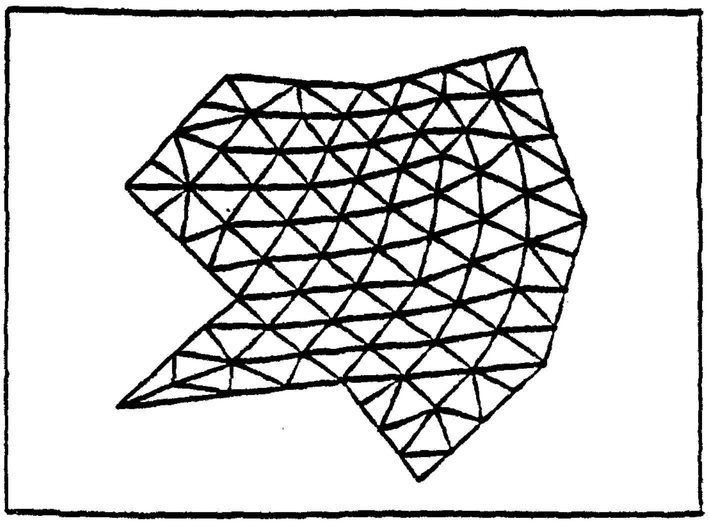
b. Grid after one refinement.



c. Grid after one refinement and one reconnection.

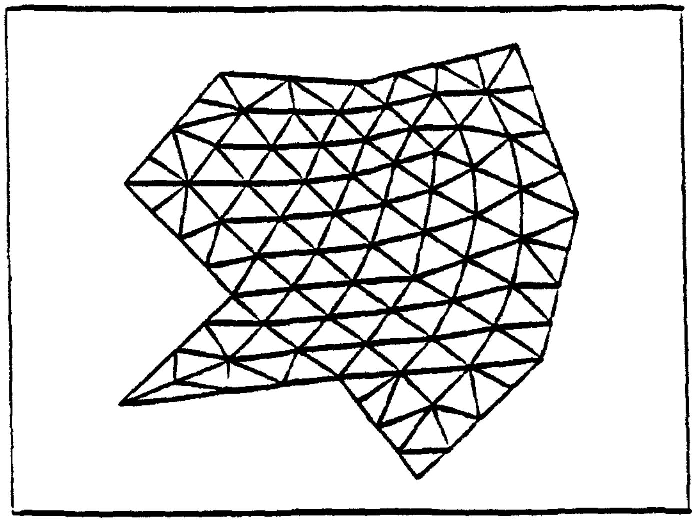


d. Second refinement.

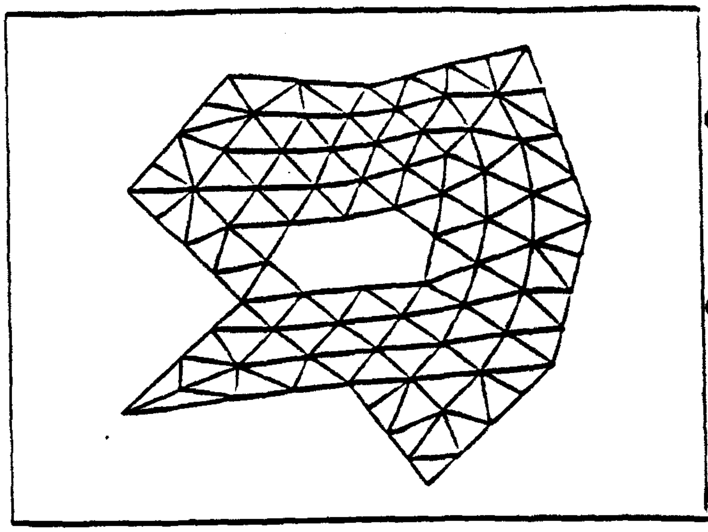


e. Second reconnection.

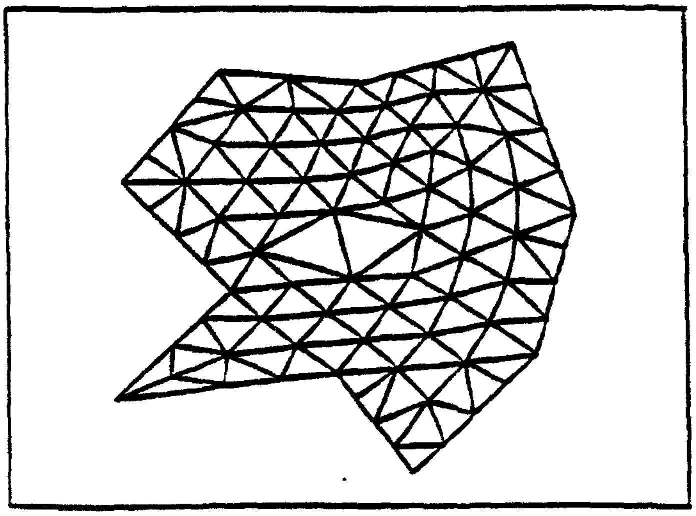
Figure 2. Illustration of the grid refinement process.



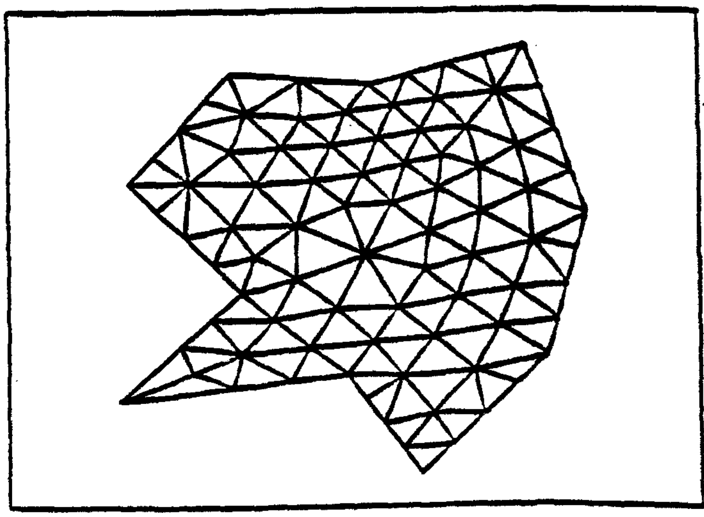
a. Original grid.



b. Point removal.



c. Constructing of new cells.



d. Grid after reconnection and relaxation.

Figure 3. Illustration of the grid coarsenning process.

No. 22

Region	ρ/ρ_0
(0)	1.00
(1)	3.70
a	1.80
b	2.56
c	3.32
d	4.08
e	4.84
f	5.60
g	6.36
h	7.12
i	6.74
j	6.36
k	7.12
l	7.50
m	7.88

Figure 4. An experimental interferogram taken at 96 μs after shock wave hits a diamond shaped obstacle, Mach M_s = 2.85.

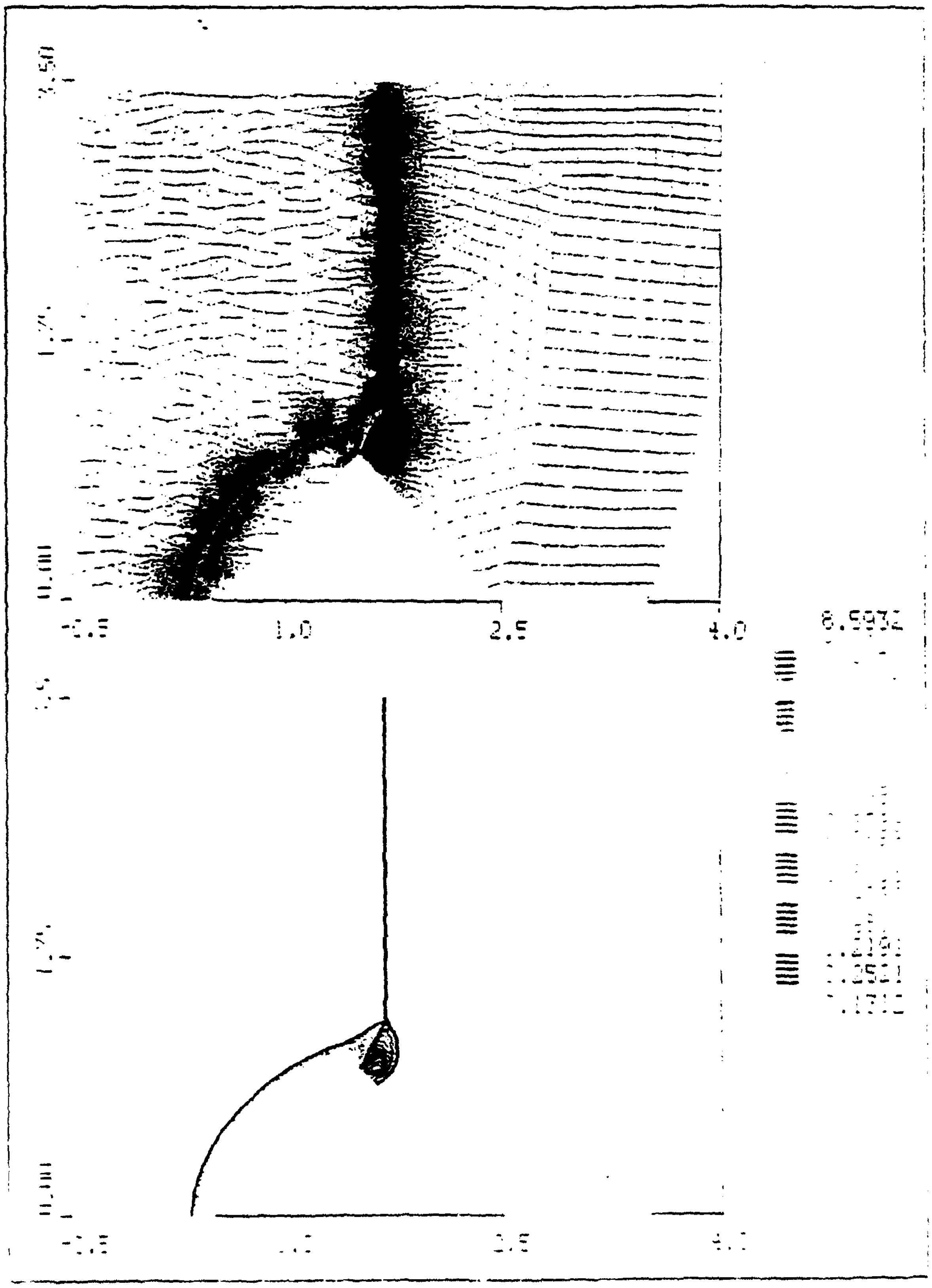


Figure 5. Computed density contours simulating flow identical to the setup of the experiment of Fig. 4. The grid is composed of 21121 vertices.

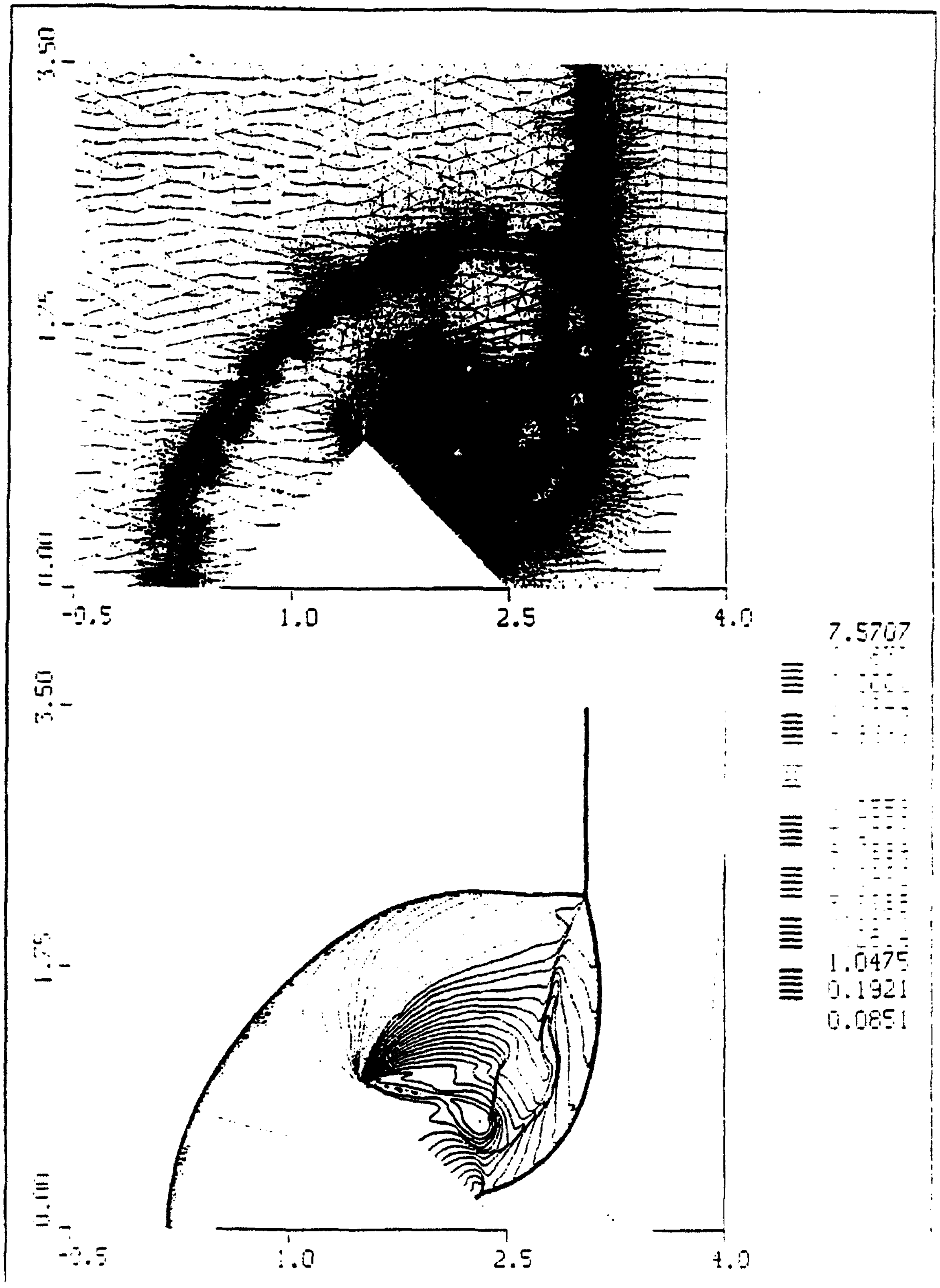


Figure 6. Computed density contours simulating flow identical to the setup of the experiment of Fig. 4. The grid is composed of 65624 vertices.

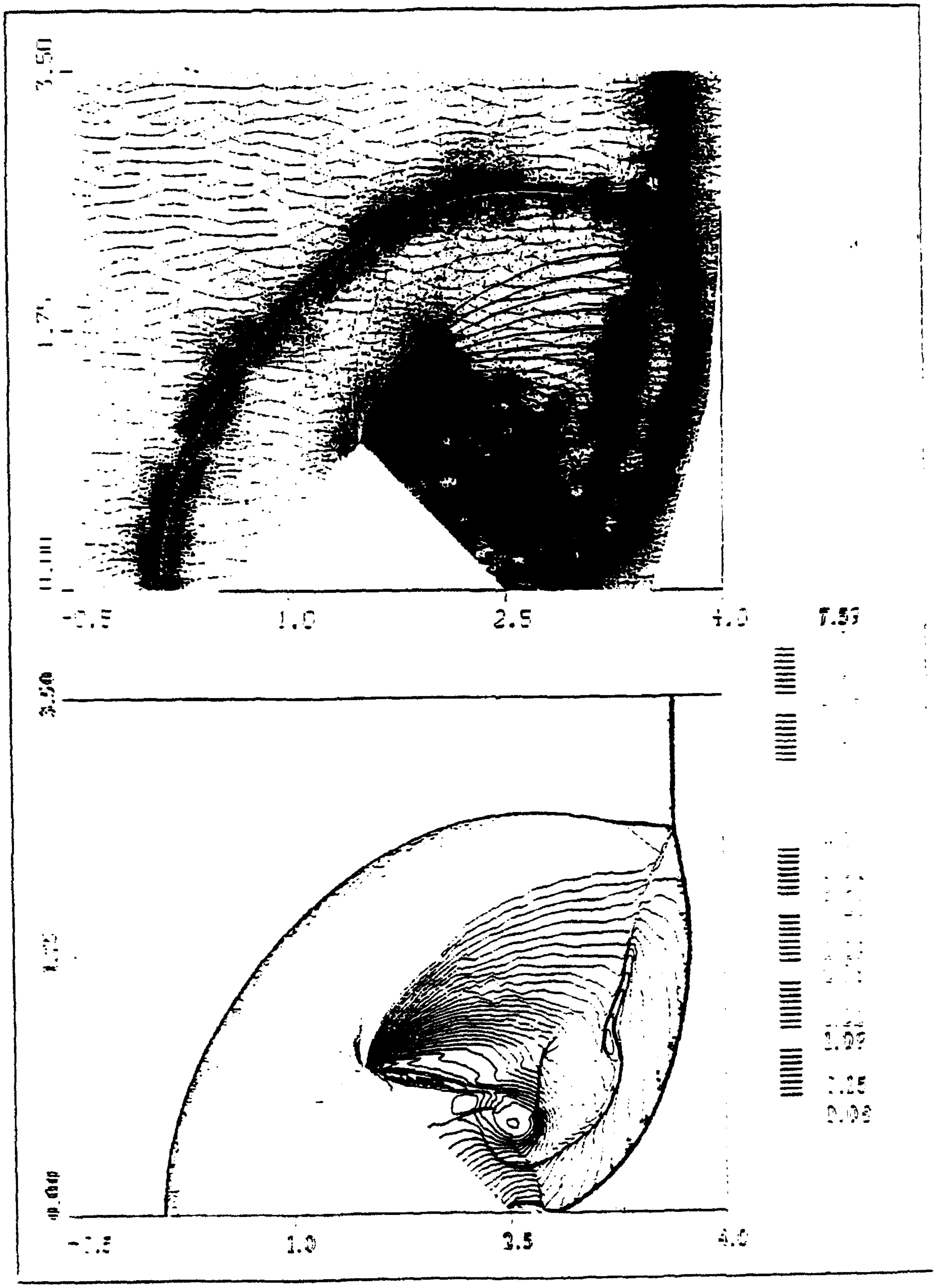


Figure 7. Computed density contours comparable to time of the experimental results shown in Fig. 4. The grid is composed of 79352 vertices.

A Second Order Godunov Scheme on Spatial Adapted Triangular Grid

Itzhak Lottati and Shmuel Eidelman

Science Applications International Corporation

ABSTRACT

Spatial adaptation procedure for the accurate and efficient solution of unsteady inviscid flow simulation is described. The adaptation procedures were developed and implemented applying a second order Godunov scheme. These procedures involve mesh enrichment/coarsening to either add/remove vertices in high/low gradient regions of the flow, respectively. The goal is to achieve solutions of high spatial accuracy at minimal computational cost. The paper describes a very effective error estimator to detect high/low activity regions of the flow to be enriched or coarsened, respectively. The error estimator is based on total energy and density fluxes into the cell combined with gradient of density. Included in the paper is a detailed description of the direct dynamic refinement method that is used for adaptation. A detailed simulation of a reflection and diffraction of multiple shock waves flowing over a diamond shape wedge is presented and compared with experimental results. The simulated results are shown to be in excellent agreement with the experiment primarily in that all the complicated features of the physics are accurately accounted for and the shock waves, slip lines, vortices are sharply captured.

INTRODUCTION

Considerable progress has been made over the past decade in developing methods for spatial adaptation of the computational meshes based on the numerical solution of the simulated physics. These methods are being developed to produce higher spatial accuracy in such simulation more efficiently. The goal of mesh adaptation is to enrich meshes locally, based on the numerical solution, in order to capture physical features of importance; in contrast to globally fine meshes, this process will minimize computer run times and memory costs. The methods of mesh adaptation can be categorized into three general classes: 1) mesh regeneration, 2) mesh movement, and 3) mesh enrichment.

The idea of mesh regeneration is systematically to identify high/low activity region in the flow and accordingly remesh those regions applying mesh generation code. This is done by assigning criteria for spatial accuracy and number of vertices. This procedure requires a mapping of the "old" flow solution into the "new" generated meshes by using one of the interpolated schemes. For the second method, mesh movement, the number of points in the computational domain remains fixed. The adaptation procedure moves vertices from low activity regions to high gradient regions to achieve a high concentration of vertices to resolve high activity regions. The movement of the points is dictated by forcing functions in the Poisson - equation in the grid generator code. The final method of spatial adaptation is mesh

related to grid adaptation of the flow variables in the area of high gradient.

enrichment. In this method, vertices are added or removed according to the spatial resolution of the physical features in the flow. The advantages of mesh enrichment over regeneration and movement are its higher degree of flexibility in being able to add points where they are needed and to remove points where they are not needed. In our mesh enrichment method, we add points ahead of the shock wave, thus preventing the need of interpolation in the high gradient region for achieving higher accuracy of the results. Adding and removing points are done in monotone/very low activity regions to prevent numerical dissipation.

Lohner[1] has developed procedures to enrich the mesh for transient flow problems locally by subdividing elements in the grid according to specific spatial resolution criteria. The method, referred to as H-refinement, keeps a history of the initial grid (mother grid) and the subdivision of each level (daughter grids). The H-refinement relies heavily on the initial grid as it is subdivided for enrichment and recovered in the coarsening stage. A similar adaptive strategy to Lohner is adopted by Rausch[2] et al., but applies a different error estimator and upwind type algorithm for a solver.

In our paper, we describe a Godunov scheme to solve Euler equations on an unstructured adaptive triangle mesh. We discuss the methodology of a cell centered Second Order Godunov scheme applied to a triangular mesh, and the method of Direct Dynamic Refinement that is used for adaptation of the unstructured triangular grid. Simulation and experimental results

are compared for a test case applying the adaptive unstructured grid to a complicated pattern of planar shock wave flow diffraction over a half diamond shape wedge.

SECOND ORDER GODUNOV ALGORITHM ON UNSTRUCTURED GRID

This section describes the implementation of the Second Order Godunov algorithm on a triangular unstructured grid. The algorithm is explicit and is cell-center based.

We consider a system of two-dimensional Euler equations written in conservation law form as:

$$\frac{\partial \bar{U}}{\partial t} + \frac{\partial \bar{F}}{\partial x} + \frac{\partial \bar{G}}{\partial y} = 0 \tag{1}$$

where

$$U = \begin{Bmatrix} \rho \\ \rho u \\ \rho v \\ e \end{Bmatrix}, F = \begin{Bmatrix} pu \\ \rho u^2 + p \\ \rho uv \\ u(e+p) \end{Bmatrix}, G = \begin{Bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ v(e+p) \end{Bmatrix}.$$

Here u, v are the x, y velocity vector components, p is the pressure, ρ is the density and e is total energy of the fluid. We assume that the fluid is an ideal gas. The total energy of gas is given by the following equation:

$$e = \frac{p}{\gamma - 1} + \frac{\rho}{2}(u^2 + v^2) \tag{2}$$

where γ is the ratio of specific heats. It is assumed that an initial distribution of the fluid parameters is given at $t = 0$, and the boundary conditions defining a unique solution are specified for the computational domain.

The system of governing equation (1) can be written in the following form:

$$\frac{\partial U}{\partial t} + \bar{\nabla} \cdot \bar{Q} = 0 \tag{3}$$

where $\bar{Q}$ represents the convective flux vector. By integrating Eq. (3) over space and using Gauss' theorem, the following expression is obtained

$$\frac{\partial}{\partial t} \int_{\Omega} U dA + \oint_{\partial\Omega} \bar{Q} \cdot d\bar{l} = 0 \tag{4}$$

where $d\bar{l} = \bar{n} d\mathcal{L}$, $\bar{n}$ is the unit normal vector in the outward direction, and $d\mathcal{L}$ is a unit length on the boundary of the domain. The variable Ω is the domain of computation and $\partial\Omega$ is the circumference boundary of this domain.

Equation (4) can be discretized for each element (cell) of the domain

$$\frac{(U_i^{n+1} - U_i^n)}{\Delta t} A_i = \sum_{j=1}^{3} \bar{Q}_j^{n+\frac{1}{2}} \bar{n}_j \Delta L_j \tag{5}$$

where A_i is the area of the cell; Δt is the marching time step; U_i^{n+1} and U_i^n are the primitive variables at the center of the cell at time n and at the update $n+1$ time step; $\bar{Q}_j$ is the value of the fluxes across the three boundaries edges on the circumference of the cell where $\bar{n}_j$ is the unit normal

vector to the boundary edge j, and ΔL_j is the length of the boundary edge j. Equation (5) is used to update the physical primitive variables U_i according to computed fluxes for each time step Δt. The time step is subjected to the CFL (Courant-Fredrichs-Lewy) constraint.

To obtain a second order spacial accuracy, the gradient of each primitive variable is computed in the baricenter of the cell. This gradient is used to define the projected values of primitive variables at the two sides of the cell's edge, as is shown in Figure 1. The gradient is approximate by a path integral

$$\int_{\Omega} \bar{\nabla} U_i^{cell} dA = \oint_{\partial\Omega} U_j^{edge} d\bar{l} \,. \tag{6}$$

The notation is similar to the one used for Eq. (5) except the domain Ω is a single cell and U_i^{cell} and U_j^{edge} are values at the baricenter and on the edge respectively. The gradient is estimated as

$$\bar{\nabla} U_i^{cell} = \frac{1}{A} \sum_{j=1}^{3} \tilde{U}_j^{edge} \bar{n}_j \Delta L_j \tag{7}$$

where $\tilde{U}_j^{edge}$ is an average value representing the primitive variable value for edge j.

The gradients that are computed at each baricenter are used to project values for the two sides of each edge by piecewise linear interpolation. The interpolated values are subjected to monotonicity constraints.[3] The monotonicity constraint assures that the interpolated values are not creating new

extrema.

The monotonicity limiter algorithm can be written in the following form:

$$U^{edge}_{projected} = U^{cell}_i + \phi \bar{\nabla} U_i \cdot \Delta \bar{r} \tag{8}$$

where $\Delta \bar{r}$ is the vector from the baricenter to the point of intersection of the edge with the line connecting the baricenters of the cells over the two sides of this edge. ϕ is the limiter coefficient that limits the gradient $\bar{\nabla} U_i$.

First, we compute the maximum and minimum values of the primitive variable in the i's cell and its three neighboring cells that share common edges (see Fig. 1):

$$\left.\begin{aligned} U^{\max}_{cell} &= Max\left(U^{cell}_k\right) \\ U^{\min}_{cell} &= Min\left(U^{cell}_k\right) \end{aligned}\right\} k = i, 1, 2, 3 \ . \tag{9}$$

The limiter can be defined as:

$$\phi = Min\left\{1, \phi^{lr}_k\right\} \;\; k = 1, 2, 3 \tag{10}$$

where superscript lr stands for left and right of the three edges (6 combinations in total). ϕ^{lr}_k is defined by:

$$\phi^{lr}_k = \frac{\left[1 + Sgn\left(\Delta U^{lr}_k\right)\right] \Delta U^{\max}_{cell} + \left[1 - Sgn\left(\Delta U^{lr}_k\right)\right] \Delta U^{\min}_{cell}}{2(\Delta U^{lr}_k)} \quad k = 1, 2, 3 \tag{11}$$

where $\Delta U_k^{lr} = \bar{\nabla} U_i^{lr} \cdot \Delta \bar{r}_k$. and

$$\left.\begin{array}{l} \Delta U_{cell}^{\max} = U_{cell}^{\max} - U_i^{cell} \\ \Delta U_{cell}^{\min} = U_{cell}^{\min} - U_i^{cell} \end{array}\right\} . \tag{12}$$

To obtain a second order of accuracy in time and space, we subject the projected values of the left and right side of the cell edge to characteristic constraints following Ref. 4. The one dimensional characteristic predictor is applied to the projected values at half time step $t^n + \frac{\Delta t}{2}$. The characteristic predictor is formulated in the local system of coordinates for the one dimensional Euler equation. We illustrate the implementation of the characteristic predictor in the direction of the unit vector $\bar{n}_c$. The Euler equations for this direction can be written in the following form:

$$W_t + A(W) W_{nc} = 0 \tag{13}$$

where

$$W = \left\{ \begin{array}{c} \tau \\ u \\ p \end{array} \right\}; \; A(W) = \begin{pmatrix} u & -\tau & 0 \\ 0 & u & \tau \\ 0 & \rho c^2 & u \end{pmatrix} \tag{14}$$

where $\tau = \rho^{-1}$, ρ denotes density while u, p are the velocity and pressure. The matrix $A(W)$ has three eigenvectors $(l^{\#}, r^{\#})$ (l for left and r for right where $\#$ denote +,o,–) associated with the eigenvalues $\lambda^+ = u + c$, $\lambda^\circ = u$, $\lambda^- = u - c$.

An approximation of projected value to an edge accurate to second order in space and time can be written as:

$$
\begin{aligned}
W_{i+\Delta r}^{n+1/2} &\approx W_i^n + \frac{\Delta t}{2}\frac{\partial W}{\partial t} + \Delta r \frac{\partial W}{\partial r_{nc}} \\
&\approx W_i^n + \left[\Delta r - \frac{\Delta t}{2} A(W_i)\right] \frac{\partial W}{\partial r_{nc}} .
\end{aligned}
\tag{15}
$$

An approximation to $W_{i+\Delta r}^{n+1/2}$ can be written as:

$$
W_{i+\Delta r}^{n+1/2} = W_i + (\Delta \bar{r}_i - \frac{\Delta t}{2}(M_x M_n) \cdot \bar{n}_c)\bar{\nabla} W_i \tag{16}
$$

where

$$
(M_x M_n) = \begin{cases} Max(\lambda_i^{+}, o) & \text{for cell left to the edge} \\ Min(\lambda_i^{-}, o) & \text{for cell right to the edge .} \end{cases} \tag{17}
$$

The gradients applied in the process of computing the projected values at $t^n + (\Delta t/2)$ are subjected to the monotonicity limiter.

Following the characteristic predictor described above, the full Riemann problem is solved at the edge. The solution of the Riemann problem defines the flux $\bar{Q}^{n+\frac{1}{2}}$ through the edge. The fluxes through the edges of triangles are then integrated (Eq. 5), thus giving an updated value of the variables at t^{n+1}. One of the advantages of the described algorithm is that calculation of the fluxes is done over the largest loop in the system (loop over edges) and can be carried out in the vectorized or parallelized loop. This fact leads to an efficient algorithm.

The algorithm presented is a modification of the algorithm of Ref. 5 which was derived for structured mesh. This algorithm has been applied to simulate a wide range of flow problems and has been found very accurate in predicting the features of the physics. The performance of the algorithm is well documented in Refs. 6-8. The next section, the spatial adaptive procedure, is described in detail. These descriptions include explanations of the error estimator for flow feature detection and the Direct Dynamic Refinement Method used to enrich and coarsen the mesh.

DIRECT DYNAMIC REFINEMENT METHOD FOR ADAPTATION ON AN UNSTRUCTURED TRIANGULAR GRID

The Direct Dynamic Refinement method (DDR) is a new method for adapting unstructured triangular grids during the computational process. As stated, an unstructured grid is very suitable for implementing boundary conditions on complex geometrical shapes as well as the adaptation of the grid, if necessary. The adaptation of the unstructured triangular grid leads to efficient usage of memory resources. The adaptive grid enables the user to capture moving shocks and high gradient flow features with high resolution. The available memory resources can be very efficiently distributed in the computational domain to accommodate the resolution needed to capture features of the physical property of the solution as they are evolved. Dynamic refinement controls the resolution priorities. These priorities can be set according to the physical features that the user wishes to emphasize

in the simulation. The user has control over the accuracy of the physical features resolved in the simulation, without being restricted to the initial grid. The alternative to Direct Dynamic Refinement (DDR) is the hierarchical dynamic refinement (H-refinement) that keeps a history of the initial grid (mother grid) and the subdivision of each level (daughter grids). In the H-refinement method, it is necessary to keep overhead information on the level of each triangle subdivision, and double indirect indexing is needed to keep track of the H-refinement process. As mentioned, the H-refinement relies heavily on the initial grid as it subdivides this grid and returns to it after the passage of the shock.

To minimize the dissipation caused by the interpolation and extrapolation in the refinement and coarsening of the grid, the addition and deletion of point is done in the region where the flow features are smooth. Thus for capturing the shock, the refinement should be applied in the region ahead of the shock. The coarsening of the grid is done in the flow regions where the gradients of the flow parameters are small.

In the present version of AUGUST (Adaptive Unstructured Godunov Upwind Second order Triangular), we implemented an algorithm with multiple criteria for capturing a variety of features that might exist in the physics of the problem to be solved. To identify the location of a moving shock, we use the flux of total energy into triangles. The fluxes entering and leaving triangles are the most accurate physical variables computed by the Godunov

algorithm for solving Euler's equations, and are used to update the physical variables for each time step in each triangle. Supplementary to the : ux of energy as an error indicator, we use the flux of total density into t · ngles and the density gradient. The error indicator is the only sensor tha: is solely responsible for identifying the area to be refined or coarsened in the computational domain. As such, the error indicator should be sensitive enough to detect physical features that are of interest to the user, such as shock waves, rarefaction waves, slip lines and vortices. The error indicators that are implemented in the code are able to sense very weak slip lines in the presence of strong shock waves. The ability of the error indicators to identify weak physical features in the presence of strong ones, without picking up numerical noises, is essential to the simulation of adaptive grids. As stated, the quality of the results is as good as the error indicators applied. If the error indicators fail to identify the physical feature, this feature probably will be overlooked in the simulated results. It should be noted that the process of applying error indicators for identifying the areas to be adaptively refined or coarsened is an expensive loop that has to check the whole triangles table in the simulation. Thus, the error indicators are applied each 9 to 15 time steps. This process is preceded by application of an algorithm that refines a buffer zone ahead of the features and coarsens the grid after it was moved away. The buffer zone ahead of the feature is identified by using a search pattern of finding the neighbors of the flagged triangles sorted by the error indicators.

We are not applying any physical parameters to identify the zones "ahead."

The refinement algorithm follows several basic steps. The process of adding points to refine the grid locally is done by either adding a new vertex in the baricenter of the triangle or adding a new vertex in the middle of the edge. Adding a new vertex in the baricenter of a triangle is very efficient in the sense that the refinement affects this individual triangle only. We apply this process exclusively for refinement. As a supplement, especially on the boundary, we apply the method of adding a new vertex on an edge. As a complement to adding new vertices, we apply the reconnection/swapping algorithm that flips the diagonal (common edge) of two adjacent triangles to improve the quality of the triangles constructed. Figure 2 displays a chain of those basic steps to illustrate the refinement process. Figure 2a shows the original grid. Figure 2b illustrates a one step scheme refinement in which a new vertex is introduced into a triangular cell forming three cells (two new ones). On the boundary edges, a new vertex is introduced in the middle of those edges to form two cells (one new one). This refinement is followed by reconnection that modifies the grid as demonstrated in Fig. 2c. The process of refinement and reconnection can be continued until the necessary grid resolution is achieved. As an example, another loop of refinement is illustrated in Figs. 2d and 2e. This direct approach to grid refinement provides extreme flexibility in resolving local flow features.

A similar direct approach is applied to grid coarsening. The basic step

in this process is deleting the cells and edges associated with a vertex to be removed, as shown in Fig. 3b. During the second step, this void in the grid is filled with new larger triangles (Fig. 3c) without introducing new vertices. The last step is local reconnection and relaxation as shown in Fig. 3d. The relaxation procedure is a simple relocation of the vertex moved to the center of the polygon surrounding this vertex (only if the polygon is a convex).

The algorithm of direct dynamic refinement proved to be very efficient in refining and coarsening the grid adaptively. The refinement and coarsening followed a short inquiry on the quality and shape of the triangle flagged and its close neighbors. Since we do not keep any history or tree for each triangle, the DDRM algorithm has much less checking to do as compared to the H-refinement algorithm. The vectorization and parallelization of the solver is straightforward.

NUMERICAL RESULTS FOR THE TWO DIMENSIONAL TEST PROBLEM

We have tested the Second Order Godunov algorithm in a variety of flow simulations ranging from the low subsonic to the high hypersonic Mach [6-8] regime. The AUGUST code proved to be very robust and accurate. The results obtained are comparable to or better than those obtained applying leading flow solvers in all of the regimes tested.

To validate our DDRM implemented in the AUGUST code, we simulated the problem of interaction of a Mach 2.85 planar shock wave, propagating

in a channel with a 45° symmetrical double ramp. Figure 4 shows the experimental interferogram of the problem to be simulated (reproduced). The example that we chose to simulate is most appropriate to test the performance of an adaptive algorithm. The experimental results show a complex flow pattern containing a mix of strong discontinuities, as shock waves, and very weak features such as slip lines, vortices, and rarefaction waves. The error estimator must recognize and flag all these features for refinement. The error estimator should be sensitive enough to identify very weak slip lines without picking up numerical noises present in the simulation. We have simulated the shock wave reflection and diffraction over a 45° corner at the conditions that correspond to the experimental result shown in Fig. 4. Here we present results for several shapes of the flow evolution. The flow in the channel is from left to right. Figure 5 displays density contour plots after the shock passed the apex of the double wedge obstacle. In Fig. 5a, the density contours are overlayed on the grid used at this stage of the evolving flow. For clarity, only the density contours are displayed in Fig. 5b. The grid displayed in Fig. 5a shows how well the adaptation technique follows the high activity region in the flow. The grid is adapting to regions with high pressure gradients and high density gradient. In Fig. 5a, one can observe high quality grid produced by the DDR method. The shock has a relatively thin buffer zone ahead of its front, allowing us to avoid the interpolations related to grid adaptation of the flow variables in the area of high gradient.

The flow features are resolved accurately, and the contact discontinuity and triple point are clearly defined.

Figure 6 shows the density contours at a later time in the same format as in Fig. 5. This figure demonstrates the ability of the DDRM to identify and follow flow features in the computational domain. In this figure we can observe a complicated flow pattern developing as a result of interaction of the rarefraction wave with the complex pattern of shock waves. A recompression shock and a strong vortex that are developed in this time frame are well resolved. We can also observe a slip line originating at the triple point. The adaptation algorithm, as in the previous time frame, follows both shock waves and contact discontinuities.

Figure 7 displays the density contours at the stage comparable to that shown in Fig. 4 for the experimental results. The computed results as displayed in Fig. 7b show a flow pattern similar to the experiment. The slip line and the formation of vertices along it are clearly depicted. The shock and reflected shock as well as the recompression shock are very sharply defined with very low numerical noise. The vortex developed after the compression shock is distinctly displayed. A new reflected shock can be seen developing at the channel wall behind the double wedge.

The results shown in Figs. 5-7 display the ability of the algorithm to simulate a complex transient flow problem on dynamically adapting grid. The error estimates used in our algorithm allow detection of strong and

weak shock waves. conducted discontinuities. vortices or other fronts that need enhanced resolution.

CONCLUSION

The Direct Dynamic Refinement (DDR) method was developed and tested for a challenging problem of reflection and diffraction of a strong shock over a double ramp. For this test problem we have demonstrated that a set of error indicators developed for the DDR allow capturing strong and weak features of the complex wave structure developing in this test case.

The above described algorithms were implemented in the AUGUST code. The AUGUST code was used for a range of subsonic. transonic. and supersonic transient and steady problems. For all these conditions the AUGUST code produced robust results with the error indicators proving to be applicable for all these diverse flow regimes.

ACKNOWLEDGMENT

The work reported here was partially supported by DARPA and AFOSR under Contract #F49620-89-C-0087. The authors would like to thank Col. James Crowley and Dr. A. Nachman for their interest in this project.

REFERENCES

1) R. Lohner, "An Adaptive Finite Element Scheme for Transient Problems in CFD," Comp. Meth. Appl. Mech. Eng., Vol. 61, 323-338. 1987.

2. R.D. Rausch. J.T. Batina. H.T.Y. Yang "Spatial Adaptation of Unstructured Meshes for Unsteady Aerodynamic Flow Computation." AIAA Journal. Vol 30. No. 5. May 1992. pp. 1243-1251.

3) B. Van Leer. "Toward the Ultimate Conservative Difference Scheme. V. A Second Order Sequel to Godunov Method." J. Comp. Phys., 32, 101-136, 1979.

4) P. Collela and H.M. Glaz. "Efficient Solution Algorithm for the Riemann Problem for Real Gases." J. Comp. Phys. 59. 264-289. 1985.

5) S. Eidelman. P. Collela, and R.P. Shreeve. "Application of the Godunov Method and its Second Order Extension to Cascade Flow Modeling," AIAA Journal 22, 10, 1984.

6) I. Lottati, S. Eidelman, and A.T. Drobot, "A Fast Unstructured Grid Second Order Godunov Solver (FUGGS)," AIAA 90-0699, 27th Aerospace Sciences Meeting, Reno, Nevada, 1989.

7) I. Lottati, S. Eidelman, and A.T. Drobot, "Solution of Euler's Equations on Adaptive Grids Using a Fast Unstructured Grid Second Order Godunov Solver," Proceedings of the Free Lagrange Conference, Jackson Lake. WY, June 1990.

8) I. Lottati and S. Eidelman, "Second Order Godunov Solver on Adaptive Unstructured Triangular Grid," Proceedings of 4th International Symposium on Computational Fluid Dynamics, Davis, CA, September 1991.

9. D.L. Zhang, I.I. Glass. "An Interferometric Investigation of the Diffraction of Planar Shock Waves Over a Half-Diamond Cylinder in Air," NTIAS Report No. 322, March 1988.

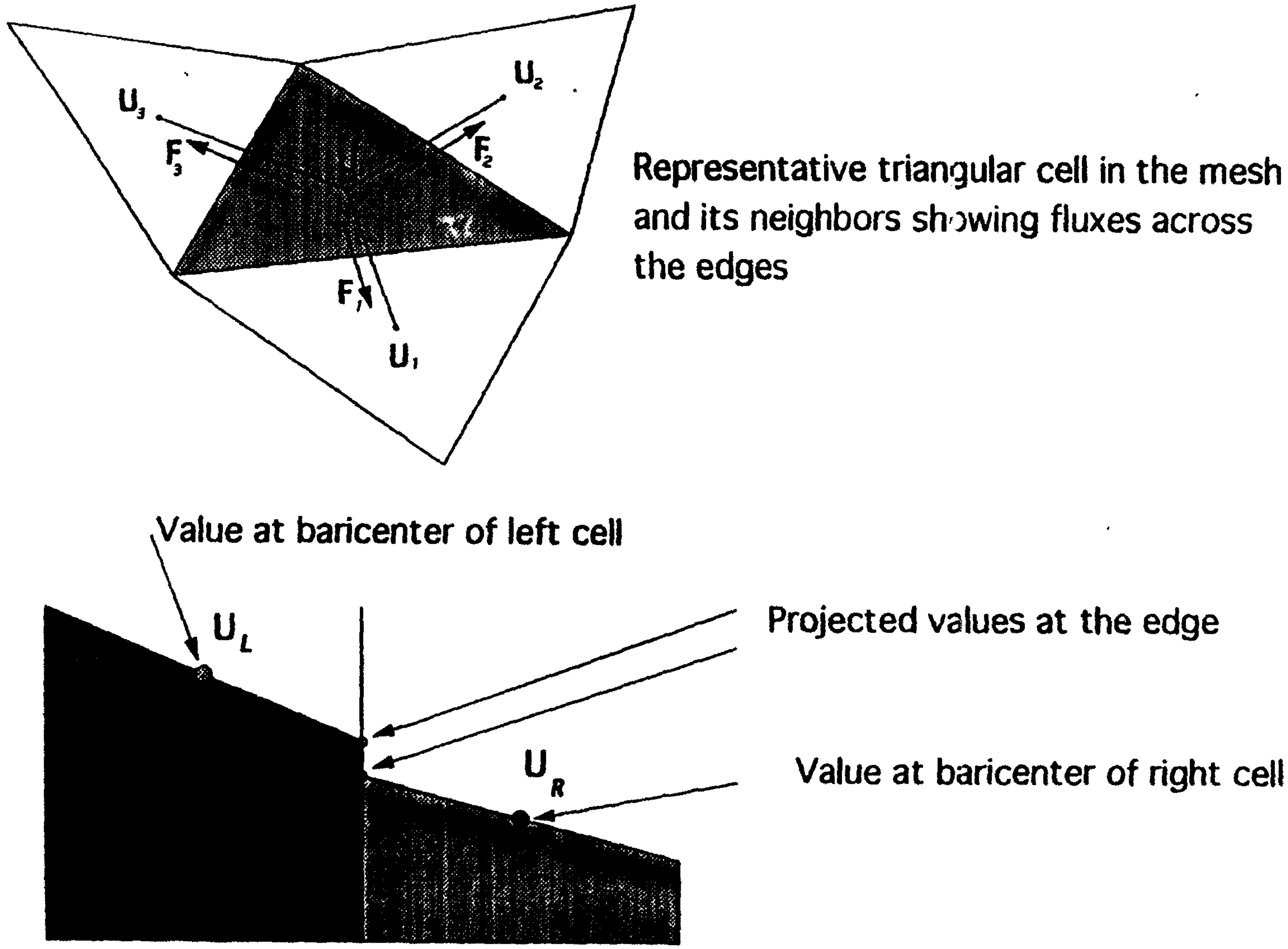
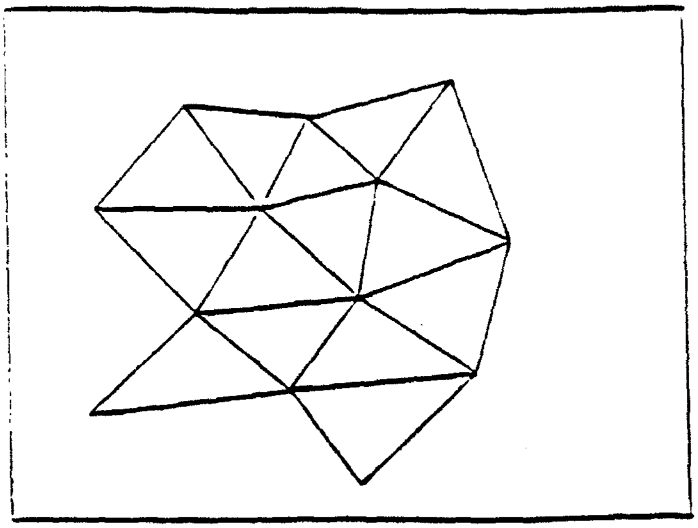
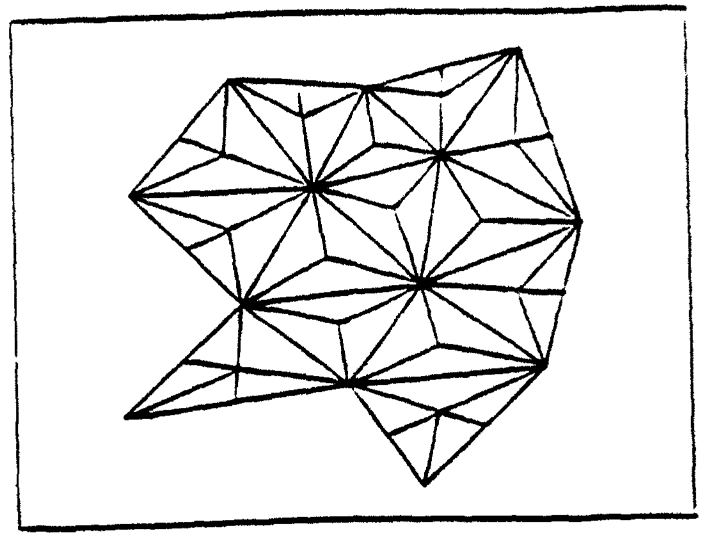


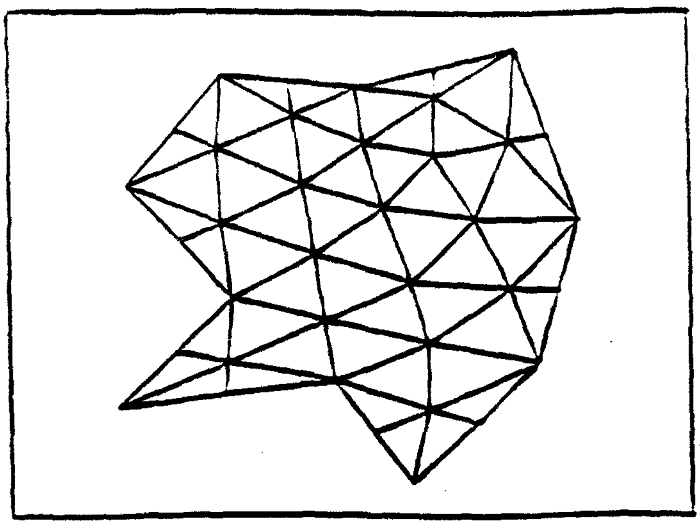
Figure 1. Representative triangular cell in the mesh showing fluxes and projected values.



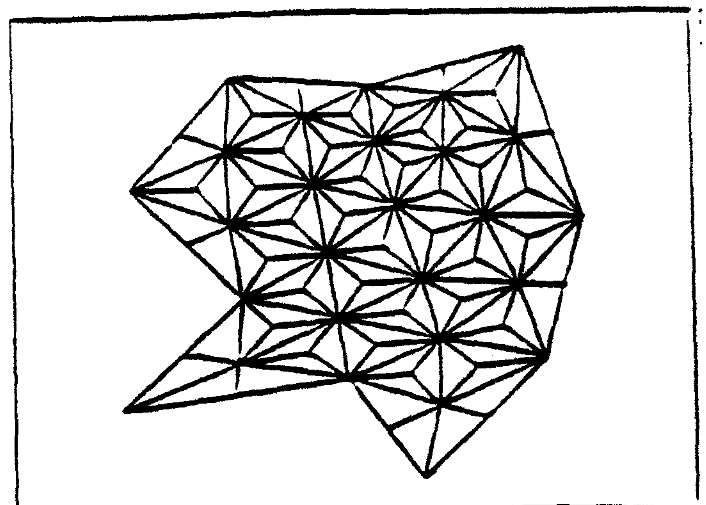
a. Original grid.



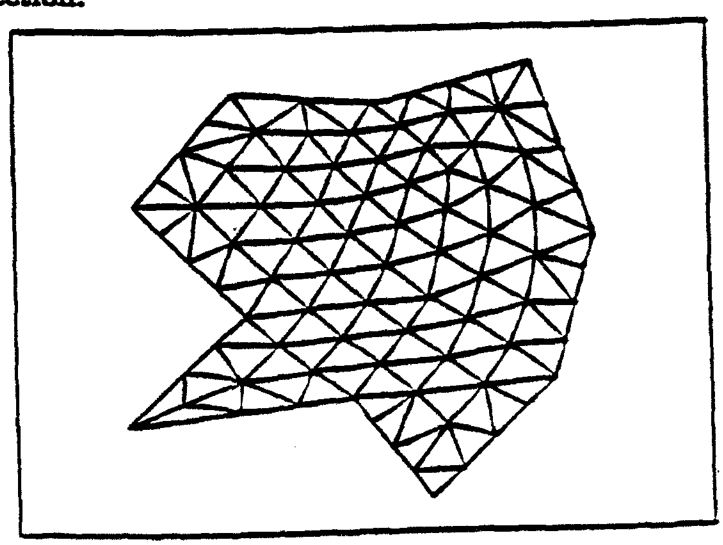
b. Grid after one refinement.



c. Grid after one refinement
and one reconnection.

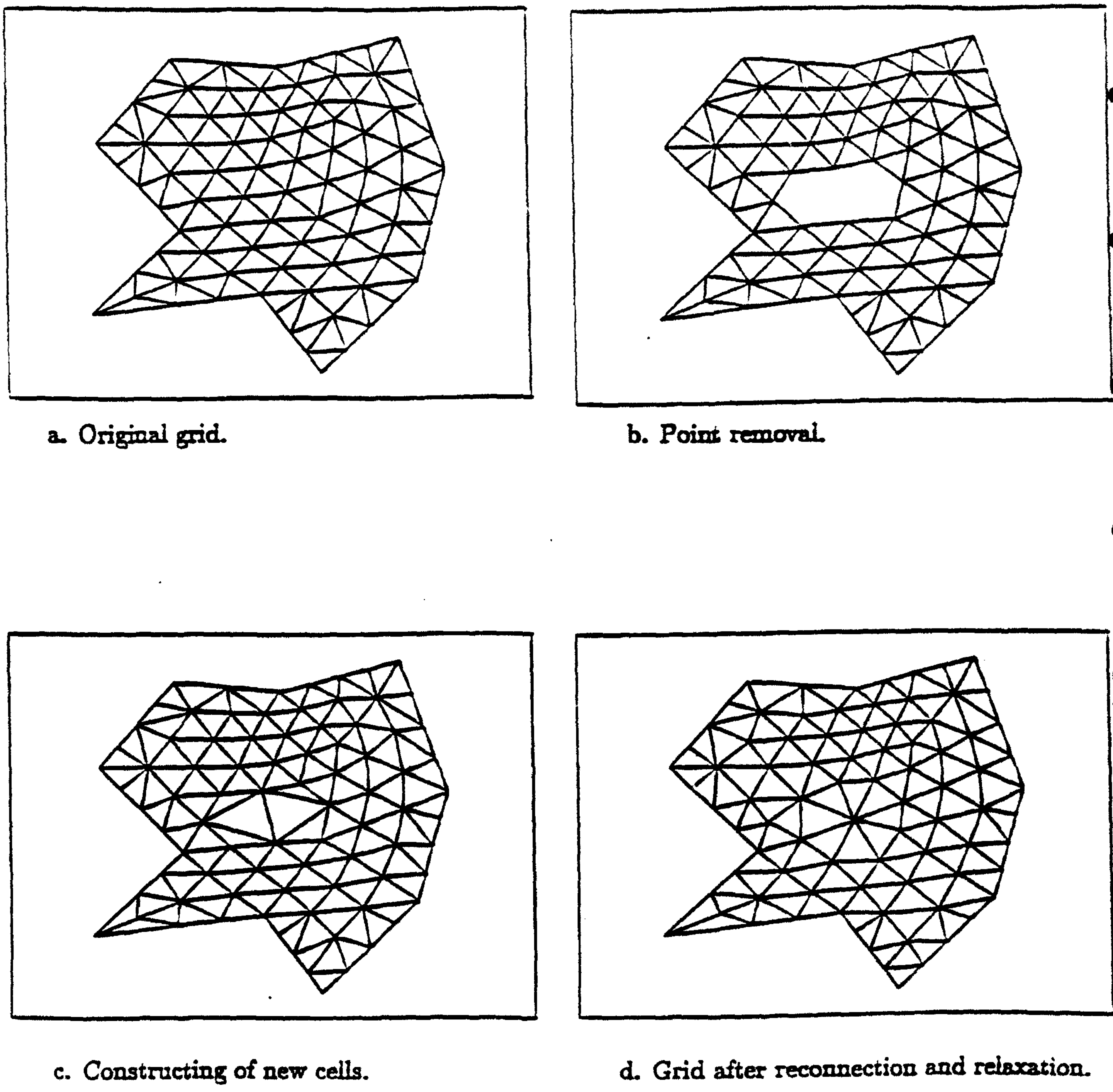


d. Second refinement.



e. Second reconnection.

Figure 2. Illustration of the grid refinement process.



a. Original grid.

b. Point removal.

c. Constructing of new cells.

d. Grid after reconnection and relaxation.

Figure 3. Illustration of the grid coarsenning process.

No. 22

Region	p/p_0
(0)	1.00
(1)	3.70
a	1.80
b	2.56
c	3.32
d	4.08
e	4.84
f	5.60
g	6.36
h	7.12
i	6.74
j	6.36
k	7.12
l	7.50
m	7.88

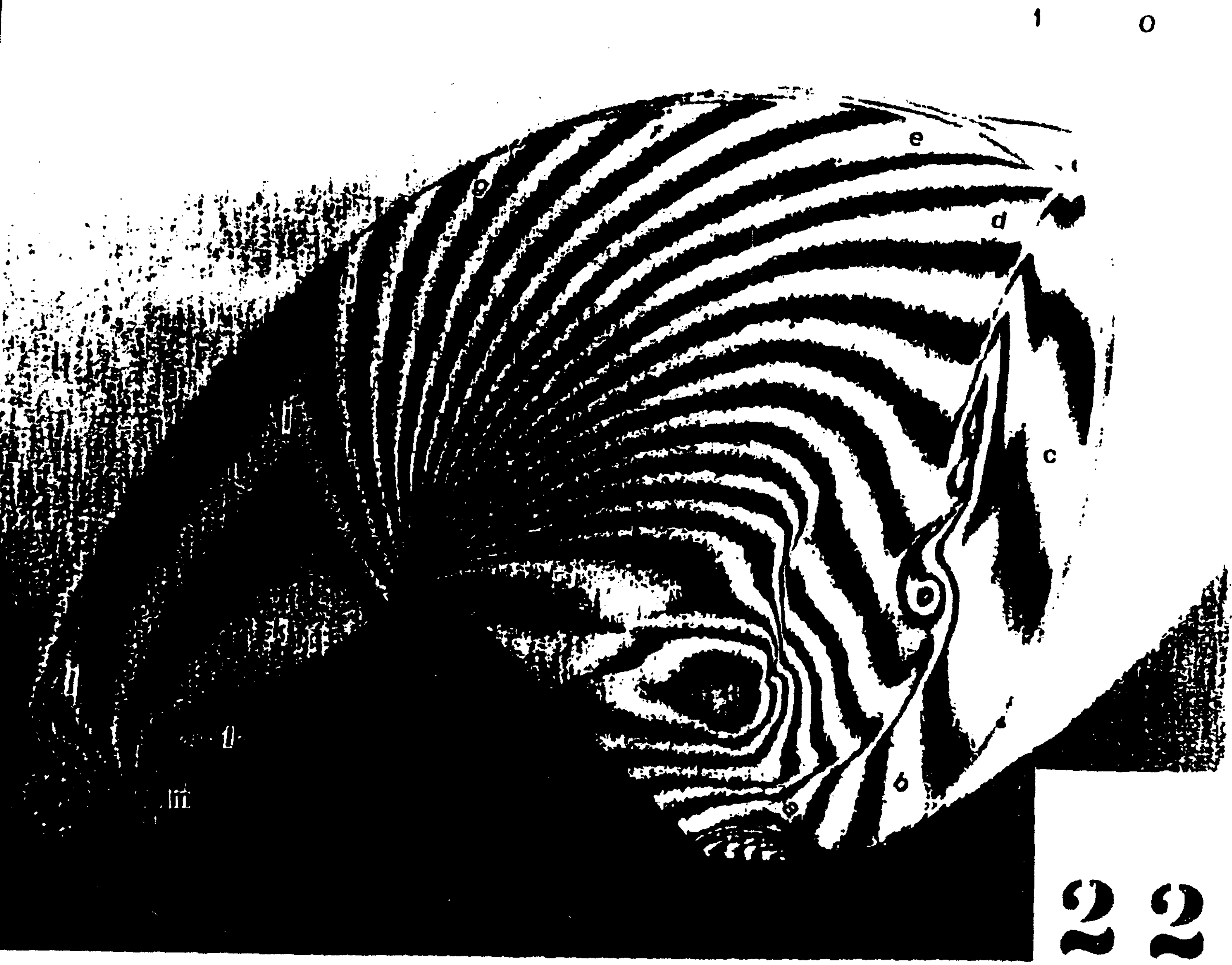


Figure 4. An experimental interferogram taken at 96 μs after shock wave hits a diamond shaped obstacle, Mach M_s = 2.85.

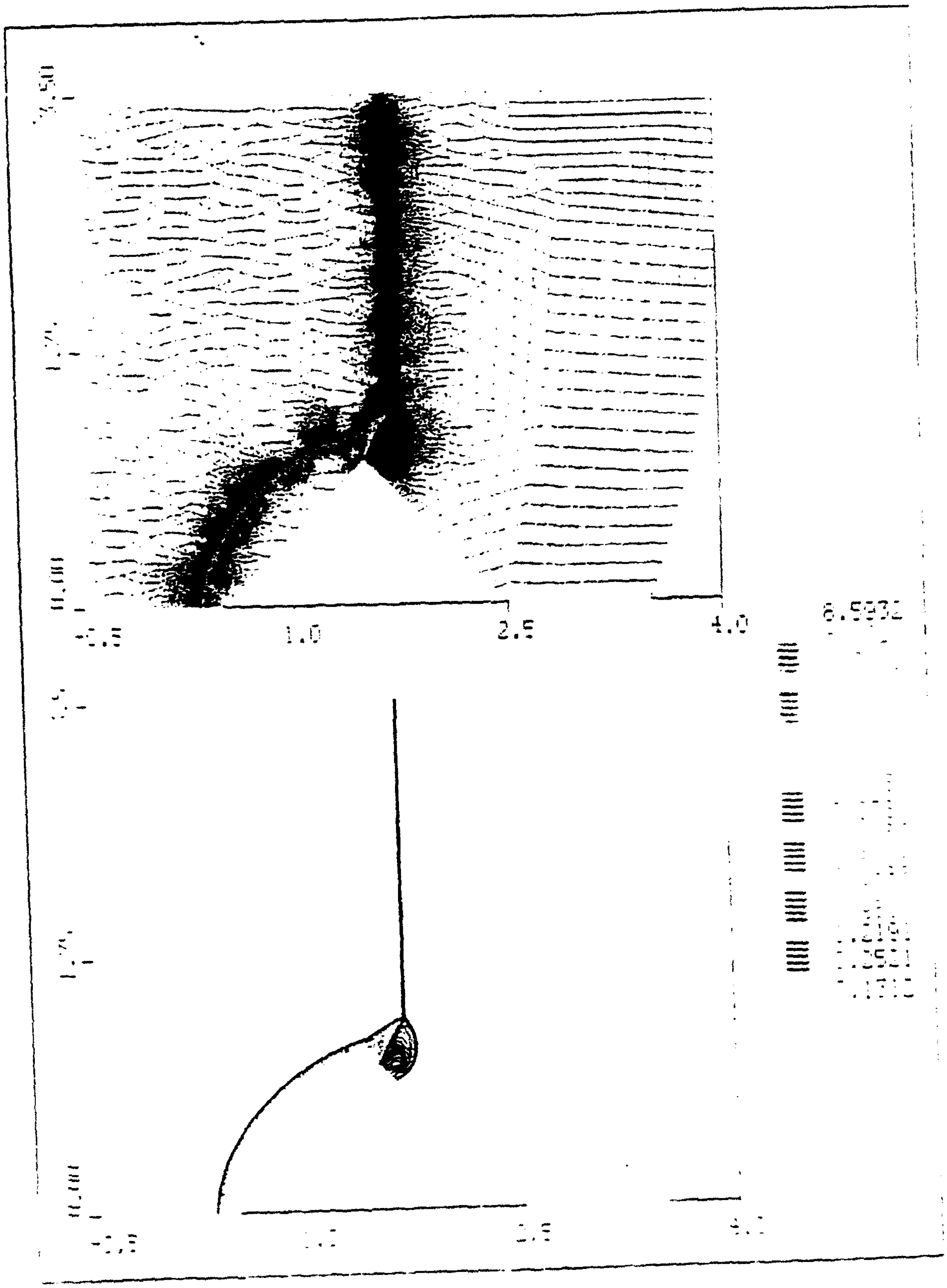


Figure 5. Computed density contours simulating flow identical to the setup of the experiment of Fig. 4. The grid is composed of 21121 vertices.

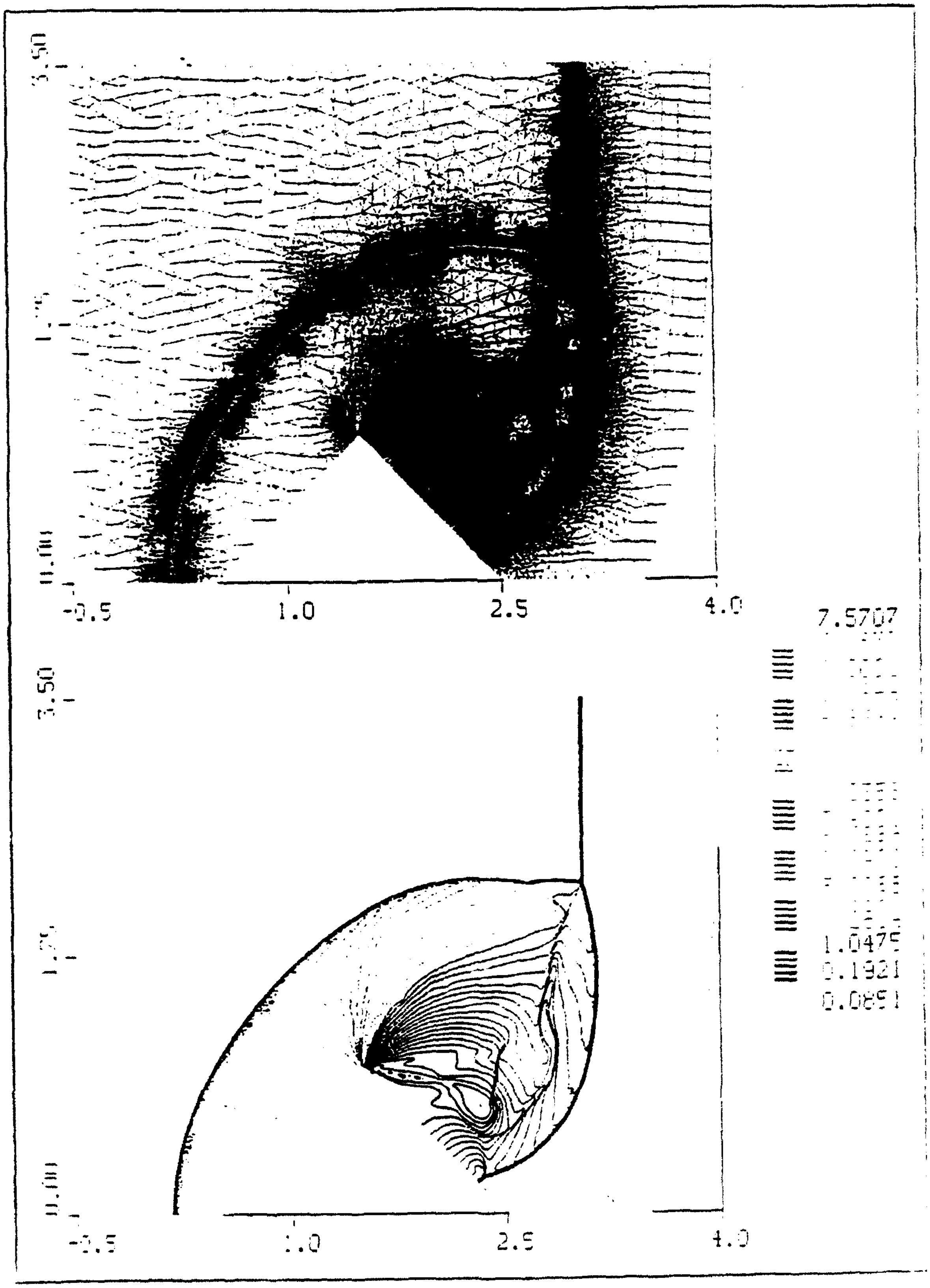


Figure 6. Computed density contours simulating flow identical to the setup of the experiment of Fig. 4. The grid is composed of 65624 vertices.

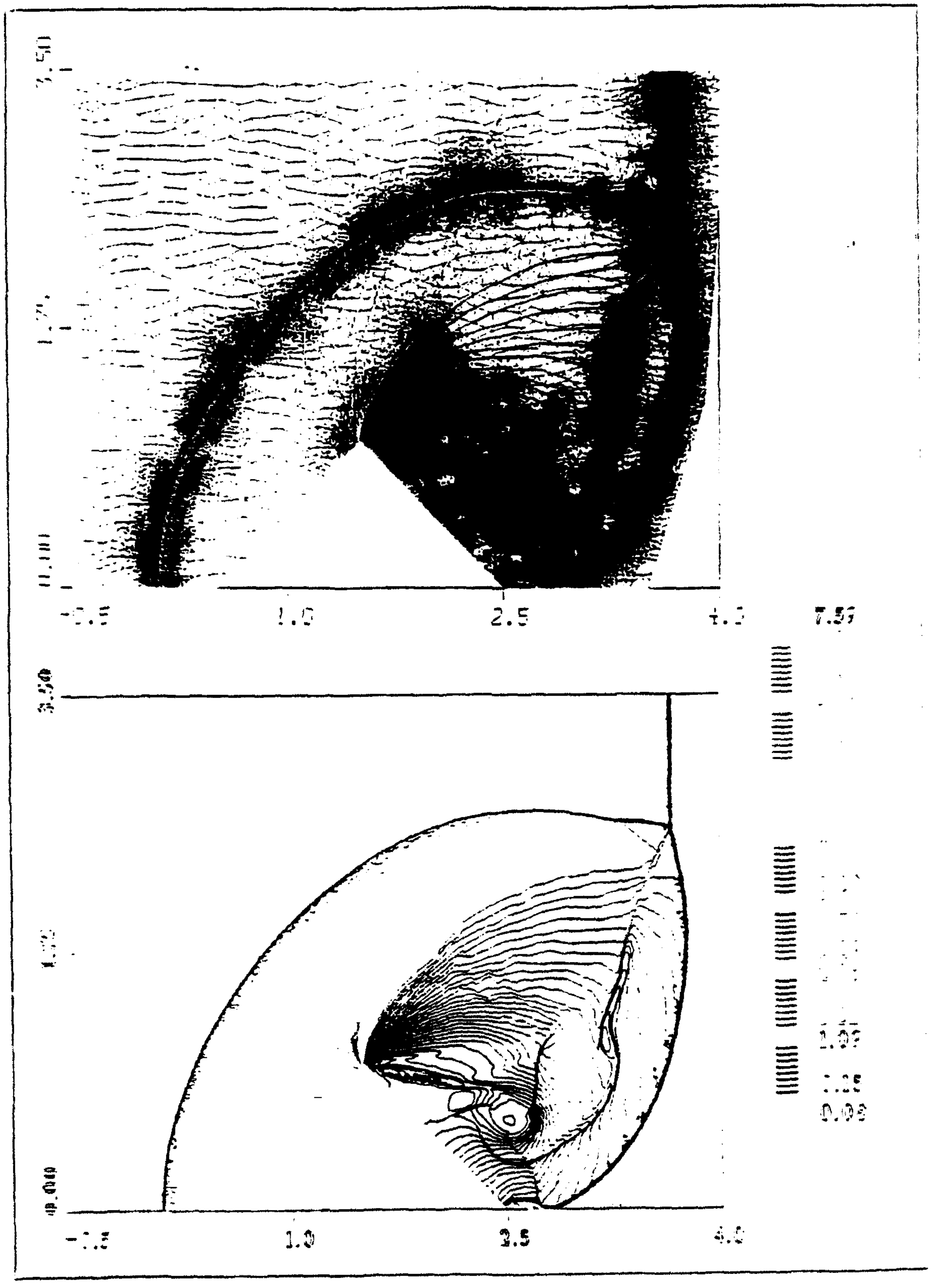


Figure 7. Computed density contours comparable to time of the experimental results shown in Fig. 4. The grid is composed of 79352 vertices.

Advances in Computer Methods for Partial Differential Equations VII
R. Vichnevetsky, D. Knight and G. Richter (Editors)

DECOMPOSITION BY STRUCTURED/UNSTRUCTURED COMPOSITE GRIDS FOR EFFICIENT INTEGRATION IN DOMAINS WITH COMPLEX GEOMETRIES

Itzhak Lottati and Shmuel Eidelman
Applied Physics Operation, MS 2-3-1
Science Applications International Corporation
McLean, VA 22102

Abstract

The Second Order Godunov method has been simultaneously implemented on both unstructured triangular and structured rectangular grids. This combined structured/unstructured method is a much more efficient approach to domain decomposition as compared to the separate application of each method. Application of this new technique to the complex problem of acoustic wave focusing in an ellipsoid reflector has demonstrated its advantages over both structured and unstructured methods of domain decomposition. It has been shown that the complex pattern of acoustic waves propagates seamlessly through structured/unstructured grid interfaces without reflection or distortion. The new approach provides ultimate flexibility in domain decomposition with minimum penalty in terms of memory and CPU requirements, and at the same time capitalizes on the advantages of both structured and unstructured grid methods.

Introduction

Structured rectangular grids allow the construction of numerical algorithms that perform an efficient and accurate integration of fluid conservation equations. The efficiency of these schemes results from the extremely low storage overhead needed for domain decomposition and the efficient and compact indexing that also defines domain connectivity. These two factors allow code construction based on a structured domain decomposition that can be highly vectorized and parallelized. Integration in physical space on orthogonal and uniform grids produces the highest possible accuracy of the numerical algorithms. The disadvantage of structured rectangular grids is that they cannot be used for decomposition of computational domains with complex geometries.

The early developers of computational methods realized that, for many important applications of Computational Fluid Dynamics (CFD), it is unacceptable to describe curved boundaries of the computational domain using the stair-step approximation available with the rectangular domain decomposition technique. To overcome this difficulty, the techniques of boundary-fitted coordinates were developed. With these techniques, the computational domain is decomposed on quadrilaterals that can be fitted to the curved domain. The solution is then obtained in the physical space using the geometrical information defining the quadrilaterals, or in the computational coordinate system that is obtained by transformation of the original domain into a rectangular domain. The advantage of this technique is that it employs the same indexing method as the rectangular structured domain decomposition methods that also serve to define domain connectivity. The boundary fitted coordinates approach leads to efficient codes, with approximately a 4:1 penalty in terms of memory requirement per cell as compared with rectangular domain decomposition. However, this approach is somewhat restricted in its domain decomposition capability, since distortion or large size variations of the quadrilaterals in one region of the domain lead to unwanted distortions or increased resolution in other parts of the domain. An example of this is the case of structured body fitted coordinates that are used for simulations of flows over a profile with sharp trailing edges. In this case, increased resolution in the vicinity of the trailing edge leads to increased resolution in the whole row of elements connected to the trailing edge elements.

The most effective methods of domain decomposition developed to overcome this disadvantage are those using unstructured triangular grids. These methods were developed to cope with very complex computational domains. The unstructured grid method, while efficient and powerful in domain decomposition, results in codes that must store large quantities of information defining the grid geometry and connectivity, and have large computational and storage overheads. As a rule, an unstructured grid code requires greater storage by a factor of 10, and will run about 5 times slower when compared on a per cell per iteration basis with a structured rectangular code.

Unstructured triangular meshes are designed to provide a grid that is fitted to the boundary of complex geometry. The flexibility of the unstructured mesh that allows gridding complex geometry should be weighed against the huge memory requirement needed to define the inter connectivity between the triangles. To cut down on the memory overhead, unstructured grid methods are used to their best advantage when combined with grid adaptivity. This feature usually allows the dynamic reallocation of triangles according to the physics and geometry of the problem solved, which leads to a substantial reduction in the number of cells needed for the domain decomposition. However, this advantage is highly dependent on the problem solved. Adaptive unstructured grids have an advantage over the unadaptive unstructured domain decomposition if the area of high resolution needed is around one-tenth of the global area of the computational domain. As a result, while the adaptive unstructured method may be extremely effective for simulating flow with multiple shock waves in complex geometries, it becomes extremely inefficient when high resolution is needed in a substantial area of the computational domain.

Our approach to domain decomposition combines the structured and unstructured methods for achieving better efficiency and accuracy. Under this method, structured rectangular grids are used to cover most of the computational domain, and unstructured triangular grids are used only to patch between the rectangular grids (Fig. 1), or to conform to the curved boundaries of the computational domain (Fig. 2). In these figures, an unstructured triangular grid is used to accurately define the curved internal or external boundaries and a structured rectangular grid is used to decompose the regions of the computational domain that have a simple geometry.

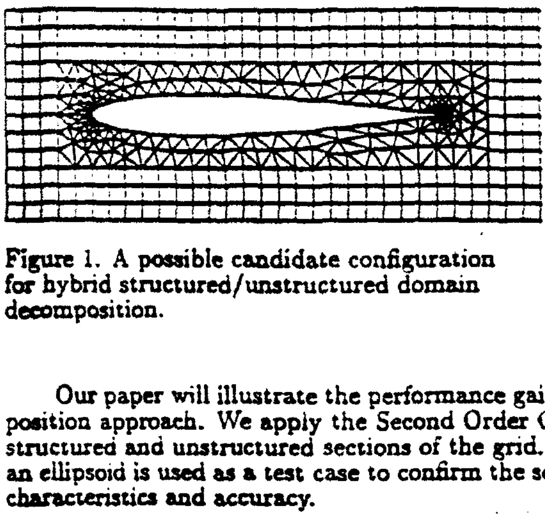


Figure 1. A possible candidate configuration for hybrid structured/unstructured domain decomposition.

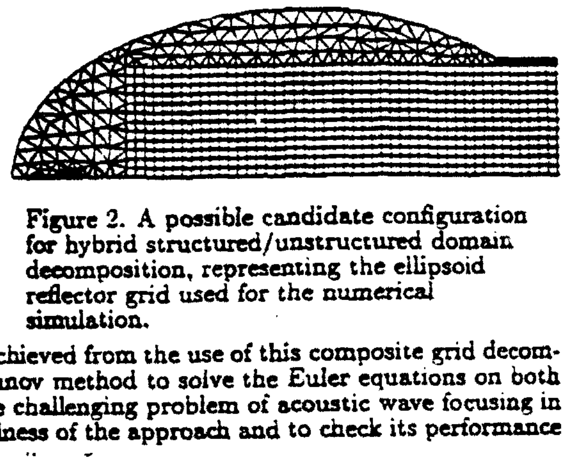


Figure 2. A possible candidate configuration for hybrid structured/unstructured domain decomposition, representing the ellipsoid reflector grid used for the numerical simulation.

Our paper will illustrate the performance gains achieved from the use of this composite grid decomposition approach. We apply the Second Order Godunov method to solve the Euler equations on both structured and unstructured sections of the grid. The challenging problem of acoustic wave focusing in an ellipsoid is used as a test case to confirm the soundness of the approach and to check its performance characteristics and accuracy.

Mathematical Model and Integration Algorithm

We consider a system of two-dimensional Euler equations written in conservation law form as:

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = 0 \tag{1}$$

where

$$U = \begin{vmatrix} \rho \\ \rho u \\ \rho v \\ e \end{vmatrix}, \quad F = \begin{vmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ u(e+p) \end{vmatrix}, G = \begin{vmatrix} \rho v \\ \rho u v \\ \rho v^2 + p \\ v(e+p) \end{vmatrix}.$$

Here u, v are the x, y velocity vector components, p is the pressure, ρ is the density and e is total energy of the fluid. We assume that the fluid is an ideal gas and the pressure is given by the equation-of-state

$$p = (\gamma - 1)(e - 0.5\rho(u^2 + v^2)) \tag{2}$$

where γ is the ratio of specific heats and typically taken as 1.4 for air. It is assumed that an initial distribution of the fluid parameters is given at $t = 0$, and the boundary conditions defining a unique solution are specified for the computational domain.

The system of governing equations in Eq. (1) can be written as

$$\frac{\partial U}{\partial t} + \nabla \cdot Q = 0 \tag{3}$$

where Q represents the convective flux vector. By integrating Eq. (3) over space and using Gauss' theorem, the following expression is obtained

$$\frac{\partial}{\partial t}\int_{\Omega} U dA + \oint_{\partial\Omega} Q \, dl = 0 \tag{4}$$

where $dl = nd\mathcal{L}, n$ is the unit normal vector in the outward direction, and $d\mathcal{L}$ is a unit length on the boundary of the domain. The variable Ω is the domain of computation and $\partial\Omega$ is the circumference boundary of this domain.

Eq. (4) can be discretized for each element (cell) in the domain

$$\frac{(U_i^{n+1} - U_i^n)}{\Delta t} A_i = \sum_{j=1}^{M} Q_j^n \, n_j \, \Delta L_j \tag{5}$$

where A_i is the area of the cell; Δt is the marching time step; U_i^{n+1} and U_i^n are the primitive variables at the center of the cell at time n and at the update $n+1$ time step; Q_j is the value of the fluxes across the M boundaries on the circumference of the cell where n_j is the unit normal vector to the boundary edge j, and ΔL_j is the length of the boundary edge j. The fluxes Q_j^n are computed applying the Second Order Godunov algorithm, and Eq. (5) is used to update the physical primitive variables U_i according to computed fluxes for each marching time step Δt. The marching time step is subjected to the CFL (Courant-Frerichs-Lewy) constraint.

We seek a solution to the system of Eq. (1) in the computational domain, which is decomposed in part into triangles with arbitrary connectivity and in part into rectangles using a logically structured grid. We use the advantage of the unstructured grid (Refs. 1-4) to describe the curved boundary of the computational domain and areas that need increased local resolution. In our example, the unstructrued grid covers 10% of the total computational domain while the structured grid occupies the remaining 90%. The numerical technique for solving Euler's equation on an unstructured grid is described in Refs. 5-7, and the technique for the structured grid is described in Ref. 8. These numerical techniques apply some of the ideas that were introduced in Refs. 9-10. The structured and unstructured codes apply the center-based formulation, i.e., the primitive variables are defined in the center of the cell, which makes the cell the integration volume, while the fluxes are computed across the edges of the cell. The basic algorithmic steps of the Second Order Godunov method can be defined as follows:

1. Find the value of the gradient at the baricenter of the cell for each gas dynamic parameter U;
2. Find the interpolated values of U at the edges of the cell using the gradient values;
3. Limit these interpolated values based on the monotonicity condition (Ref. 9);
4. Subject the projected values to the characteristic's constraints (Ref. 10);
5. Solve the Riemann problem applying the projected values at the two sides of the edges;
6. Update the gas dynamic parameter U according to the conservation equations (1) applying to the fluxes computed and the current time step.

As was advocated in Ref. 7, we prefer the triangle center-based over the vertex-based version of the code. For the same unstructured grid, a triangle-based algorithm will result in smaller control volumes than a vertex-based. In addition, for the Second Order Godunov solver, implementation of the boundary conditions is more straightforward and accurate for the center-based algorithm than in the vertex-based. These two factors, along with the effects of grid connectivity, strongly affect the algorithm accuracy and

performance, and are the main reasons for the superiority of the center-based version over the vertex version.

Sound Wave Focusing in an Ellipsoid Reflector

Research relating to focusing of shock and acoustic waves is of considerable practical interest for application to Extracorporeal Shock Wave Lithotripsy (ESWL). Most of the interest in this area is related to acoustic waves in liquids: however, the basic reflection and focusing mechanisms for a given reflector geometry can be studied in air as well. For our test simulation, we chose a deep reflector shaped like an ellipsoid, which was used for ESWL by Dornier (Ref. 11) and other companies. A schematic of the cross section of this reflector is shown in Fig. 3. Strong acoustic waves are generated in the left focal point of the ellipsoid by an instantaneous release of energy and are refocused at the right focal point. Ideally, focusing should be based on waves of acoustic intensity, since the nonlinear reflections of strong shock waves lead to significant distortions in wave propagation and impair simple geometrical focusing.

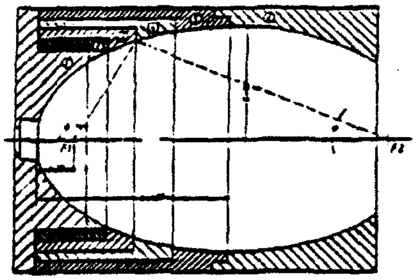


Figure 3. A schematic drawing of the center cross section of the ellipsoid reflector.

Figure 2 shows the computational domain and grid for the ellipsoid reflector example. In order to illustrate the concept of the composite structured/unstructured grid, we have shown only every 1/16 cell of the grid that was actually used for the simulation. In this example, we observe that the structured rectangular grid covers about 90% of the computational domain and the unstructured triangular grid is restricted to the curved surface of the ellipsoid and covers about 10% of the domain. The major axis of the ellipsoid is 150 mm and the minor axis is 90 mm.

The integration in the structured part of the domain is performed using a version of the split Second Order Godunov method described in Ref. 8. For the unstructured triangular grid, we used our implementation of the Second Order Godunov method that includes a compact integration stencil suitable for unstructured grids (Refs. 5-7). In the current implementation, the two sections of the grid communciate through the boundary conditions at their interfaces. According to this, the values in the mirror points at the grid interfaces for the triangular grid are taken from the computational domain of the structured grid and vice versa. These mirror values are used for calculations of the flux at the interface boundaries. For focusing problem simulations, we used 55188 triangles in the unstructured part of the grid and 141312 (736 × 192) rectangles in the structured part. It should be mentioned that in order to obtain a uniform grid (i.e., the structured and unstructured grids have the same level of refinement), the unstructured portion of the code was run with adaptivity (adding and deleting vertices). This ability enabled us to match the grid resolution based on cell areas in the structured/unstructured grids while computing the results. The initial grid had a very refined grid at the left focal point to initiate accurately the detonation. This area was coarsened later in the simulation by turning on the adaptive capability of the unstructured code.

We used the following initial condition at the time $t = 0$ for the simulation of the acoustic wave focusing:

a. Quiescent air in the cavity of the reflector, i.e., Pressure $P_o = 101350.$ Pa and Density $\rho_o = 1.2\,\mathrm{Kg/m^3}$.

b. Blast in the left focal point of the ellipsoid confined in a spherical volume of a radius of R = 2mm. Condition at initial blast area: Pressure $P_b = 45. * P_o$, and Density $\rho_b = 4.5 * \rho_o$.

This definition of the initial conditions guarantees that a weak blast wave will be generated, ensuring that waves of acoustic intensity will be reflected from the wall of the ellipsoid. We examined this particular reflection regime because the blast wave focusing in water occurs in acoustic mode. As it was pointed out in Ref. 11, reflection of even very weak waves in water will lead to considerable deviations from the reflection mode of a pure acoustic wave. However, the purpose of this simulation

is to demonstrate the numerical method and not to study in detail the focusing modes of the ellipsoid reflector. Therefore, we present results for one simulation following conditions outlined above.

In Fig. 4a, the simulation results are shown in the form of pressure contours at the time $t = 1.31 \times 10^{-4}$sec when the incident shock started its reflection from the reflector wall. Here we can observe that the maximum reflected pressure is no higher than 14% over the ambient pressure, which is consistent with our objective to create weak waves. Figure 4b is an enlargement of the region in the computational domain that contains structure and unstructured grids. We can also observe that the incident wave propagates seamlessly through the interface of the structured and unstructured regions. In Fig. 5, we show pressure contour plots at time $t = 2.09 \times 10^{-4}$sec. We observe that the interfaces between the two grids carry the information seamlessly.

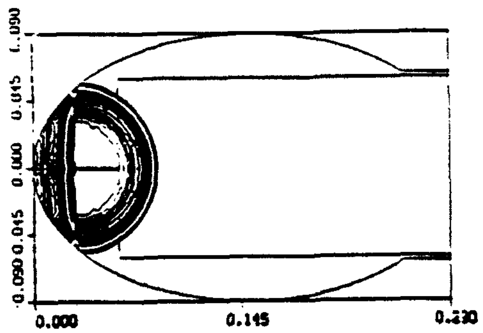


Figure 4a. Pressure contours at time $t = 1.31 \times 10^{-4}$sec showing the incident wave as reflected from the reflector's wall.

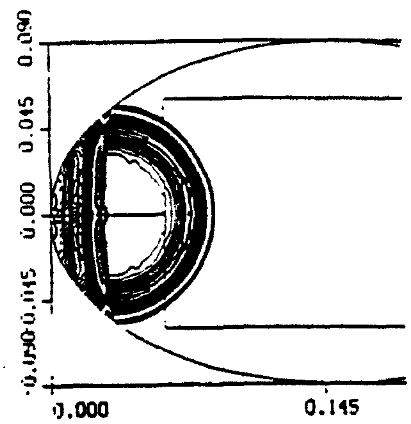


Figure 4b. Blowup of the pressure contours at time $t = 1.31 \times 10^{-4}$sec showing the matching pressure contours between the structured and the unstructured grid.

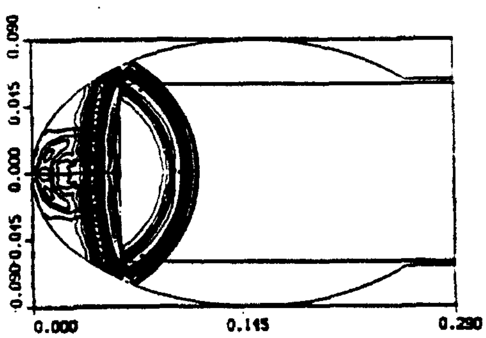


Figure 5. Pressure contours at time $t=2.09 \times 10^{-4}$sec showing the incident wave and the reflected wave pattern.

Figure 6 shows the simulation results at time $t = 4.35 \times 10^{-4}$sec. At this stage, the blast wave front that propagated to the left has undergone full reflection and the reflected wave propagates in the direction of the incident wave to the right. However, the incident and the reflected wave are both of acoustic intensity and they are propagating at the speed of sound. Therefore, the reflected wave will not be able to catch up with the incident wave at this stage of expansion. We can observe in Fig. 7, where the two waves are shown past the ellipsoid centers ($t = 5.41 \times 10^{-4}$sec), that the distance between these acoustic waves does not change as compared with Fig. 6. The reflected wave has maximum pressure in the vicinity of the axis and its value remains relatively constant (about 1.10×10^5 Pa) through the propagation process. The wave complex at the axis of symmetry consists of the incident acoustic wave front, a reflected wave that has positive followed by negative phases.

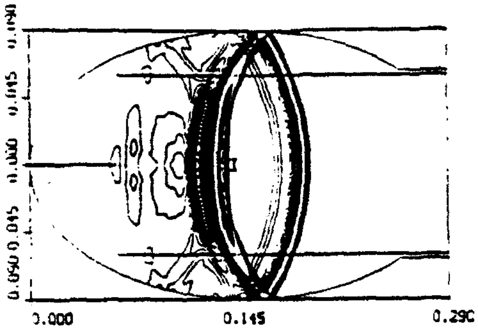


Figure 6. Pressure contours at time $t=4.35 \times 10^{-4}$sec showing the incident wave and the reflected wave pattern.

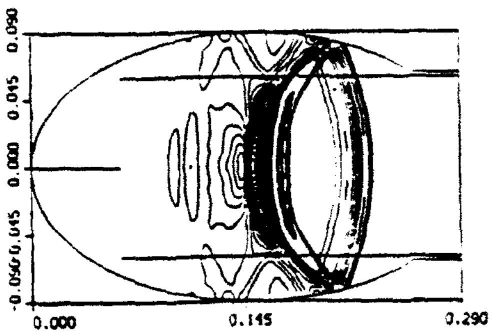


Figure 7. Pressure contours at time $t=5.41 \times 10^{-4}$sec showing the wave pattern past the center of the ellipsoid.

The enhancement of the reflected wave's amplitude starts gradually when the reflected wave is approaching the second focal point caused by the convergence of the ellipsoid. In Fig. 8, the pressure contours ($t = 8.41 \times 10^{-4}$sec) are shown at the stage that the maximum focused pressure is obtained in the system. As we can observe in Fig. 8, the incident front has left the computational domain, and the maximum pressure is obtained in small volume in the vicinity of the right focal point. In our simulation, the maximum focused pressure has reached 1.32×10^{5}Pa and is located 11 mm to the right of the focal point of the ellipsoid.

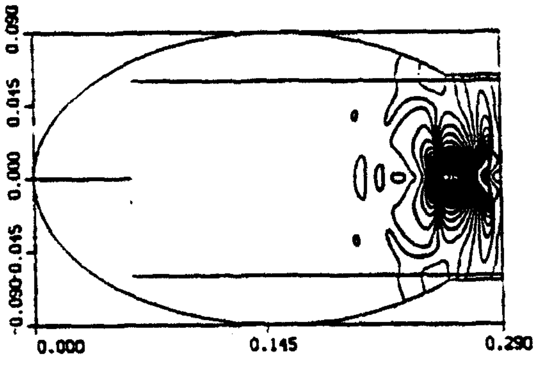


Figure 8. Pressure contours at time $t=8.41 \times 10^{-4}$sec showing the stage at which the maximum focused pressure is obtained.

In all the figures presented, the method of composite domain decomposition works extremely well, producing seamless solutions at the interfaces. We should mention here that our test problem is particularly sensitive because the main acoustic waves are weak, and any inaccuracy introduced at the grid interfaces would produce a distortion in the phase or in the intensity of the traveling waves that would be a visible disturbance evident in the results needless to mention that an adaptive scheme would have difficulty in simulating this problem due to the weakness of the wave pattern.

Conclusions

A composite method of structured/unstructured domain decomposition is introduced as an efficient technique for dealing with the computational domains of complex geometry. We have simulated a demanding acoustic wave focusing problem and have shown that our approach leads to accurate wave propagation without any reflection or distortion at the structured/unstructured grid interfaces. It should be noted that for the acoustic focusing problem as simulated and presented in this paper, both structured and unstructured methods of domain decomposition can be shown to be inadequate if used separately. The structured method has difficulty describing the curved boundaries of the computational domain, while the unstructured method is totally inefficient in describing phenomena with wide fronts that occupy a large portion of the computational domain. Our hybrid method combines the advantages of structured and unstructured methods of domain decomposition. This hybrid technique combines the efficiency of the unstructured grid to accurately represent curved walls, with the computational and memory efficiency of the structured grid in the majority of the computational domain. We also attribute the quality of the numerical result to the Second Order Godunov method, which allows a consistent, accurate and robust formulation for handling both grids and boundary conditions.

Acknowledgments

The work reported here was partially supported by DARPA and AFOSR under Contract #F49620-89-C-0087. The authors would like to thank Col. James Crowley and Dr. A. Nachman for their interest in this project.

References

1. A. Jameson, T.J. Baker, and N.P. Weatherill, "Calculation of Inviscid Transonic Flow Over a Complete Aircraft," AIAA 24th Aerospace Sciences Meeting, Reno, NV, AIAA Paper 86-0103, January 1986.
2. R. Löhner, "Adaptive Remeshing for Transient Problems," Comp. Meth. Appl. Mech. Eng. 75, 195-214, 1989.
3. J. Peraire, M. Vahdati, K. Morgan, and O.C. Zienkiewicz, "Adaptive Remeshing for Compressible Flow Computations," J. Comp. Phys. 72, 449-466, 1987.
4. D. Mavriplis, "Accurate Multigrid Solution of the Euler Equations on Unstructured and Adaptive Meshes," AIAA 88-3707, 1988.
5. I. Lottati, S. Eidelman, and A. Drobot, "A Fast Unstructured Grid Second Order Godunov Solver (FUGGS)," 28th Aerospace Sciences Meeting, AIAA-90-0699, Reno, NV, 1990.
6. I. Lottati, S. Eidelman, and A. Drobot, "Solution of Euler's Equations on Adaptive Grids Using a Fast Unstructured Grid Second Order Godunov Solver," Proceeding of the Free Lagrange Conference, Jackson Lake, WY, June 1990.
7. I. Lottati and S. Eidelman, "Second Order Godunov Solver on Adaptive Unstructured Grids," Proceeding of the 4th International Symposium on Computational Fluid Dynamics, Davis, CA, September 1991.
8. S. Eidelman, P. Collela, and R.P. Shreeve, "Application of the Godunov Method and Its Second Order Extension to Cascade Flow Modeling," AIAA Journal 22, 10, 1984.
9. B. van Leer, "Towards the Ultimate Conservative Difference Scheme, V: A Second Order Sequel to Godunov's Method," J. Comp. Phys. 32, 101-136, 1979.
10. P. Collela and P. Woodward, "The Piecewise Parabolic Method (PPM) for Gasdynamic Simulations," J. Comp. Phys. 54, 174-201, 1984.
11. H. Gronig, "Past, Present and Future of the Shock Focusing Research," Proceedings of the International Workshop on Shock Wave Focusing, Sendai, Japan, March 1989.

Advances in Computer Methods for Partial Differential Equations VII
R. Vichnevetsky, D. Knight and G. Richter (Editors)

TWO-PHASE COMPRESSIBLE FLOW COMPUTATION ON ADAPTIVE UNSTRUCTURED GRID USING UPWIND SCHEMES

Xiaolong Yang, Shmuel Eidelman and Itzhak Lottati
Applied Physics Operation, MS 2-3-1
Science Applications International Corporation
McLean, VA 22102

ABSTRACT

A computer program called MPHASE for numerical study of shock wave propagation in a multiphase, multi-component gas environment is described and applied. The mathematical model of the multiphase, multi-component system is based on the multi-fluid Eulerian approach. Basically, we consider the two phases(i.e. gas and particle) to be interpenetrating continua: the dynamics of the flow is governed by conservation equations for each phase, and the two phases are coupled by interactive drag force and heat transfer. The code is formulated on unstructured triangular grids.

The numerical solution method is based on the Second Order Godunov Method for the gaseous medium, an upwind integration for the particles, and an implicit integration technique for the gas-particle interaction simulation. In order to produce a solution with high spatial accuracy at minimal computational cost, an adaptive procedure on the unstructured grid is used. The adaptive procedure will automatically enrich the grid by adding points in the high-gradient (or high flow activity) region and by removing points (coarsening the mesh) where they are not needed. This technique allows a detailed study of the complex two-phase shock reflection phenomena, where the effects of momentum and heat exchange between phases will significantly modify the shock structure and shock parameters.

Results will be given from the code validation study for the shock propagation in the dusty gases. The code performance will be illustrated by solving the problem of reflection and diffraction of a plan shock wave over a semicircular cylinder in a dusty gas.

1. THE MATHEMATICAL MODEL AND THE NUMERICAL SOLUTION

Conservation Equations

The mathematical model consists of conservation governing equations and constitutive laws that provide closure for the model. The basic formulation adopted here follows the gas and dilute particle flow dynamics model presented by Soo[1]. The following assumptions are used during the derivation of governing equations:

(1) The gas is air and is assumed to be ideal gas;

(2) The particles do not undergo a phase change because particles are considered as sand whose phase transition temperature is much higher than the gas temperature considered here;

(3) The particles are solid spheres of uniform diameter and have a constant material density;

(4) The volume occupied by the particles is negligible;

(5) The interaction between particles can be ignored;

(6) The only force acting on the particles is drag force and the only heat transfer between the two phases is convection. The weight of the solid particles and their buoyancy force are negligibly small compared to the drag force;

(7) The particles have a constant specific heat and are assumed to have a uniform temperature distribution inside each particle.

Under the above assumptions, distinct equations of continuity, momentum, and energy are written for each phase. The interaction effects between the two phases are listed as the source terms on the righthand side of the governing equation. The two dimensional unsteady conservation equations for the two phases can be written in the vector form in Cartesian coordinates:

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = S. \tag{1}$$

Here U is the vector of conservative variables, F and G are fluxes in x and y direction, respectively, and S is the source term for momentum and heat exchange. The definition of these vectors are:

$$U = \begin{vmatrix} \rho \\ \rho u \\ \rho v \\ e \\ \rho_p \\ \rho_p u_p \\ \rho_p v_p \\ e_p \end{vmatrix}, \quad F = \begin{vmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ u(e+p) \\ \rho_p u_p \\ \rho_p u_p^2 \\ \rho_p u_p v_p \\ u e_p \end{vmatrix}, \quad G = \begin{vmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ v(e+p) \\ \rho_p v_p \\ \rho_p u_p v_p \\ \rho_p v_p^2 \\ v_p e_p \end{vmatrix}, \quad S = \begin{vmatrix} 0 \\ -f_x \\ -f_y \\ -q - u_p f_x - v_p f_y \\ 0 \\ f_x \\ f_y \\ q + u_p f_x + v_p f_y \end{vmatrix}$$

where ρ, u, v, and e are gas density, velocities, and energy, respectively; ρ_p, u_p, v_p and e_p are particle density, velocities, and energy, respectively; (f_x, f_y) and q denotes drag force components acting on the particles and heat transfer to the particles, respectively. The gas pressure p is related to ρ, u, v and e for by

$$p = (\gamma - 1)[e - 0.5\rho(u^2 + v^2)] \tag{2}$$

where γ is the specific heat ratio. The gas temperature can be found through the equation-of-state for ideal gas

$$p = \rho RT \tag{3}$$

where R is the gas constant.

The particle temperature T_p is calculated through relation

$$e_p = \rho_p c_p T_p + 0.5\rho_p(u_p^2 + v_p^2). \tag{4}$$

The source terms on the righthand side of equation (1) are momentum and heat exchange between gas and particle phases. If we let r_p and ρ_s be the particle radius and material density, respectively, then the drag forces are

$$\begin{pmatrix} f_x \\ f_y \end{pmatrix} = \frac{3}{8}\,\frac{\rho_p \rho}{\rho_s r_p}\, C_d \left[(u - u_p)^2 + (v - v_p)^2\right]^{1/2} \begin{bmatrix} u - u_p) \\ (v - v_P) \end{bmatrix}. \tag{5}$$

The particle drag coefficient C_d is a function of Reynolds number, Re, which is based on the relative velocity between the gas and particle phases. After testing the drag coefficients given by Sommerfeld[2] and by Clift *et al.*[3], the following were two adopted:

$$C_d = \frac{24}{Re}(1 + 0.15Re^{0.687}) \text{ for } Re < 800$$

and (5)

$$C_d = \frac{24}{Re}(1 + 0.15Re^{0.687}) + \frac{0.42}{1 + 42500Re^{-1.16}} \text{ for } Re > 800\,.$$

Here the Reynolds number, Re is defined as

$$Re = \frac{2\rho r_p[(u - u_p)^2 + (v - v_p)^2]^{1/2}}{\mu} \tag{6}$$

Viscosity, μ is calculated at film temperature, namely, $T_f = 0.5(T_p + T)$, and the temperature dependency of the viscosity is evaluated according to Sutherland's law

$$\mu = \mu_r \left(\frac{T}{T_r}\right)^{3/2} \frac{T_r + \Phi}{T + \Phi} \tag{7}$$

where μ_r is the dynamic viscosity of the gaseous phase at the reference temperature and Φ is an effective temperature, called the Sutherland constant.

The rate of heat transfer from gaseous phase to the particle phase is given by

$$Q = \frac{3}{2} \frac{\rho_p}{\rho_s} \frac{\mu C_p}{Pr} Nu \, (T_o - T_p) \tag{8}$$

where $Pr = \mu c_p / k_g$ is the Prandtl number, and c_p and k_g are the specific heat and thermal conductivity of gas, respectively. The Nusselt number Nu is a function of this Reynolds number and the Prandtl number as given by Drake[4]

$$Nu = \frac{2r_p h}{R} = 2 + 0.459 Re^{0.55} Pr^{0.33}. \tag{9}$$

Initial and Boundary Conditions

The geometry of the computational domain is shown in Fig. 1. The initial conditions for gas are $\rho_o = 1.2 kg/m^3$ and $p_o = 101.3 kpa$, with a coming shock at x = -0.5. There are no particles from - 1.0 $\leq$ x $\leq$ 0.0. From $x \geq 0.0$, particles are initially in thermal and kinematic equilibrium with surrounding gas. The particles that are uniformly distributed in the dusty region have the following parameters for different test problems:

Mass loading, ρ_p: 0.25 kg/m^3, 0.76 kg/m^3;

Mass material density, ρ_s: 2500 kg/m^3;

Particle radii, r_p: 10 μm, 25 μm, 50 μm;

Specific heat, c_s: 766 $J/kg/K$.

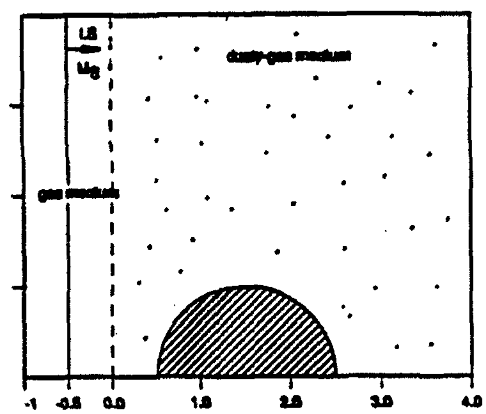


Figure 1. An illustration of the considered flow field.

The lower boundary and cylinder surface are solid walls and assumed adiabatic and impermissible. A reflecting boundary condition is assumed for both the gas and particle phase. Particles are assumed to experience a perfect elastic collision with the wall and reflect from the wall. The right and upper boundaries are open boundaries where a nonreflection boundary condition is used for the gas phase and a zero normal gradient condition is used for particle phase.

Numerical Method of Solutions

The system of partial differential equations described in the previous paragraph is integrated numerically. Equation (1) is repeated here:

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = S. \tag{1}$$

In order to solve this equation numerically, an operator time-splitting technique is used. Assuming that all flow variables are known at a given time, we can calculate its advancement in time by splitting the integration into two stages.

In the first stage, the conservative part of equation (1) is solved:

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = 0. \tag{10}$$

The Second Order Godunov method is used for the integration of the subsystem of equations describing the flow of the gaseous phase (first four components of equation (1)). The method is well documented in literature.[5,6,7] The subsystem of equations describing the particle phase flow is integrated using a simple finite difference upwind scheme. This is done because there is no shock in the particle phase and the upwind scheme leads to a robust and accurate integration scheme.

In the second stage, the source term is added and the following equation is solved:

$$\frac{\partial U}{\partial t} = S. \tag{11}$$

To integrate this equation in time, we need to obtain S as a function of U. We calculate S through equations (5) to (8).

In order to produce a solution of the high spatial accuracy at minimal computational cost, an unstructured triangular grid with adaptive procedure is used. The adaptive procedure will automatically enrich the mesh by adding points in the high gradient (or high flow activity) region of the flow field and by removing points (coarsening mesh) where they are not needed. The dynamic nature of mesh enrichment is shown in Fig. 3 for two different time frames. One can see that a very fine mesh is generated around shock fronts and other steep density gradient regions.

2. RESULTS

Model Validation for One-Dimensional Shock Wave Propagation in A Dusty Gas

To test the momentum and heat exchange mechanism for the current two-phase model, we first simulate a one-dimensional problem of a normal shock wave propagating into a dusty gas. We numerically simulate the experiments conducted by Sommerfeld[2]. In the experiments, small glass sphere particles of material density $\rho_s = 2500 kg/m^3$, specific heat capacity $c_s = 766 J/kg/K$, and average diameter of 27 μm were used as suspension particle phase. The incoming shock, and particle loading ratio $\eta = \rho/\rho_p$, are two varying parameters. The experimental results and our numerical simulation results of shock Mach number as a function of distance for two test cases are shown in Fig. 2a ($\eta = 0.63$ and Fig. 2b ($\eta = 1.4$) for comparison purpose. As one can see, the agreement between the prediction of our present model and the experimental results is very good.

Two-Dimensional Simulation Results of Pure Gas Flow

To test the accuracy of the two-dimensional computation, we compute the pure gas flow case of a shock wave reflection and diffraction over a semicircular cylinder. We then compare the simulation with experimental results. Shock wave reflection on a wedge has been extensively studied by many researchers (see e.g., review paper of Hornung[8]). Shock wave reflection by circular cylinders was numerically simulated by Yang *et al.*[9] and experiments were performed by Kaca[10]. Fig. 3a and 3b show density contours with adapted grids at two moments in time. In Figs. 4a nd 4b, the interferogram from the experiment[3] and density contours from the present simulation are compared for the same flow condition and same time. Note that the density levels are normalized by the ambient gas density in Fig. 4. As one can see from Fig. 4a and Fig. 4b, the results show an excellent quantitative as well as qualitative agreement between the numerical simulation and experimental results.

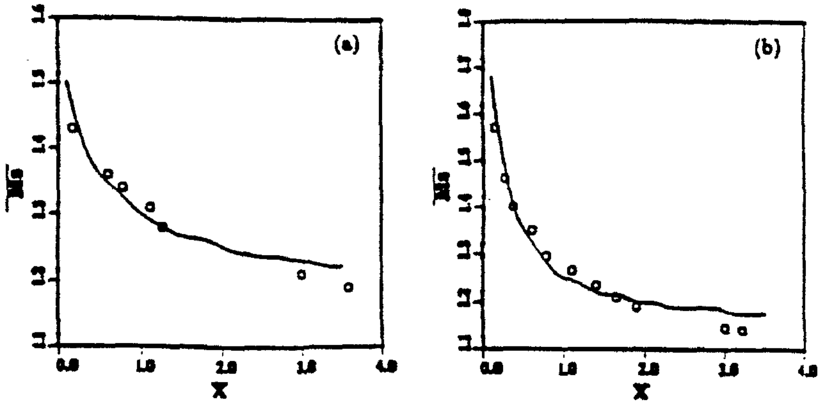


Figure 2. Comparison between computational prediction and experimental measurement of shock wave attentuation for (a) $M_o = 1.40$, $\eta = \frac{\rho_2}{\rho_o} = 0.63$ and (b) $M_o = 1.7$, $\eta = \frac{\rho_2}{\rho_o} = 1.4$ (o experiment, - calculztion).

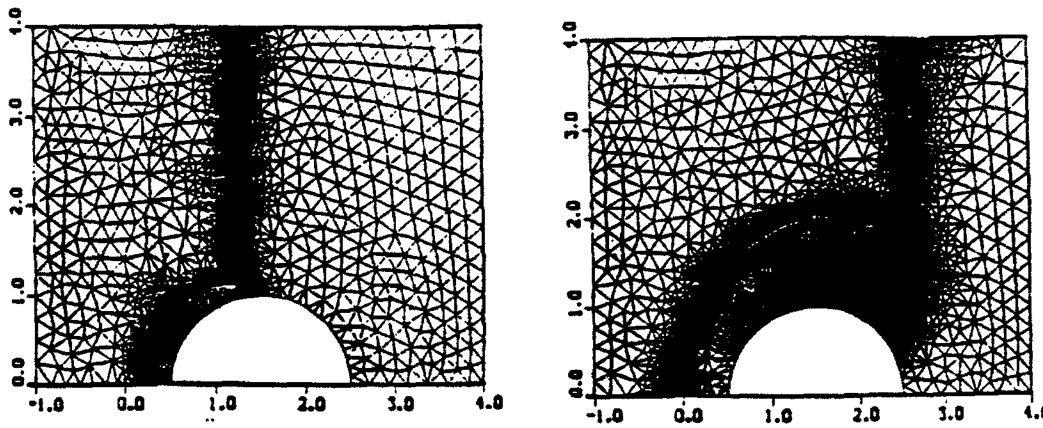


Figure 3. Computed density contours with adapted grid at two different times.

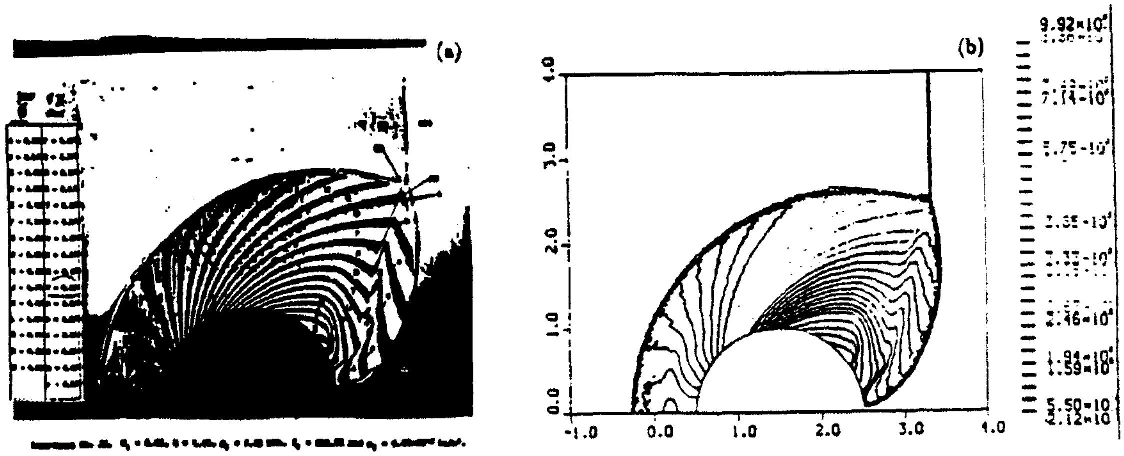


Figure 4. Comparison for $M_s = 2.80$ gas – only flow, (a) interferogram from experiment conducted by Kaca (1988), (b) density contours from present calculation.

Two-Dimensional Simulation Results of Two-Phase Flow

The basic setup for the two-phase simulation is shown in Fig. 1. Here the planar shock with Ms=2.8 impinges on an area of a dusty gas. The interface between clear air and dusty air is located at x=0.0 of the computational domain. The area of the dusty air contains a semicylinder with a radius of $1m$. The size of the computational domain, initial parameters of the gas, parameters of the incoming shock, size of the semicylinder and its location in the computational domain, are the same as in the reflection and diffraction simulation presented in the previous section.

The main objective of this set of simulations is to study the effects of particle size and particle loading on the parameters of the reflected and diffracted shock waves. It is also of interest to study the dynamics of reflection and diffraction in particle media. This is especially valuable since it is extremely difficult to observe these interactions experimentally in an optically thick dusty gas.

The first set of simulation results is shown for the case with dust parameters $r_p = 10\mu m$ and $\rho_p = 0.25\ kg/m^3$. The gas parameters and the parameters of the incoming shock wave are the same as in the pure gas case presented above. In Figs. 5a and 5b, particle density contours and gas density contours are shown at the stage when the incident shock wave has reached the top of the semicylinder. At this stage, particles have very little effect on the dynamics and parameters of the shock in the gas phase. The presence of the particles causes a small widening of the shock that is more noticeable for the incident shock. Also, one can observe an additional contour line at the dusty gas/pure gas interface. The particle density contours depict significant piling up of the dust particles at the leading edge stagnation point of the cylinder.

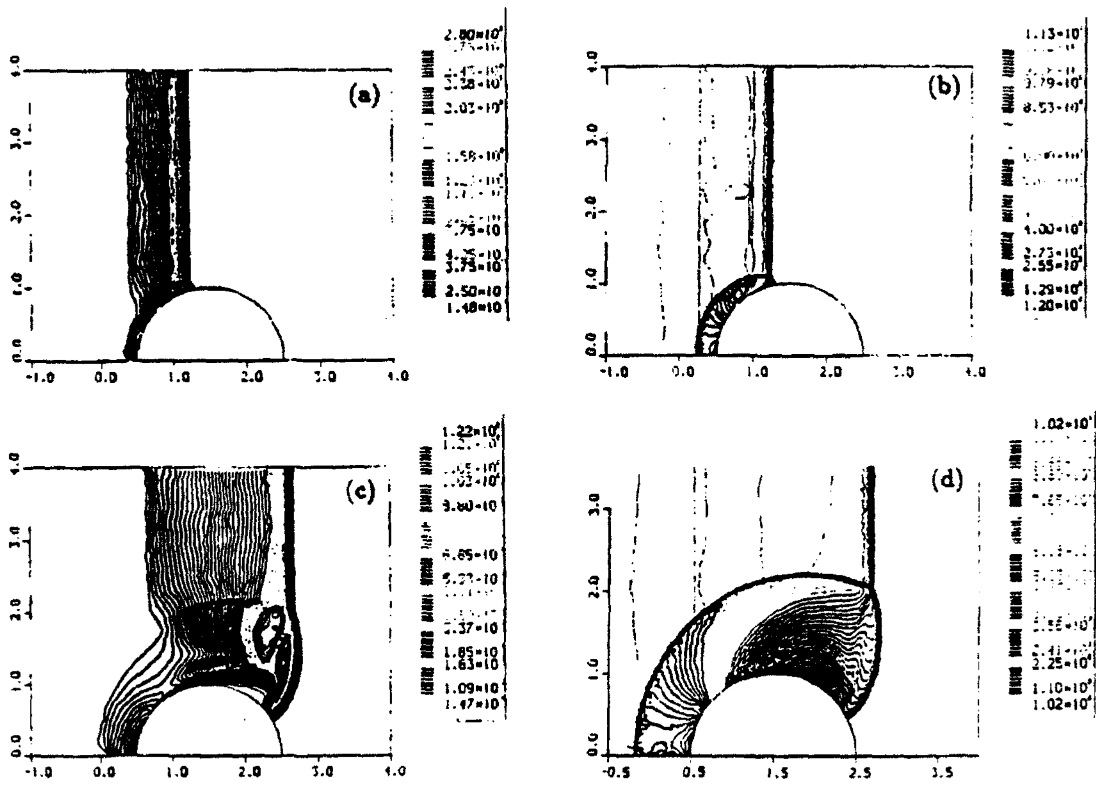


Figure 5. Density contours for the case; $M_s = 2.8$, $\rho_p = 0.25 kg/m^3, r_p = 10\mu m$ at two different times. (a) particle density at t_1, (b) gas density at t_1, (c) particle density at t_2, and (d) gas density at t_2.

In Figs. 5c and 5d, the particle density and gas density contours are shown at the stage where significant diffraction has taken place and the shock front is approaching the trailing edge of the cylinder. The small particle loading and small particle size leads to very small modification of the gas shock structure and parameters. One can observe further widening of the shock and some smearing of the slip

line that originates at the triple point. The particle density contours reveal that the particles piled up at the stagnation point were swept by the gas flow to the area of triple point and slip line for the gas flow, leaving a small amount of particles at the leading edge. We should note that this behavior is specific for our problem, where at t=0, the dusty gas area was located at x=0 and there is no influx of the dust from the left boundary. Also in Fig. 5c, we note that the particles reach a distinct local maxima at the distance about 25 *cm* behind the main shock front. At this maxima the particle density is 0.86 kg/m^3, which is more than three times the initial particle density. The particle density reaches a maximum value at the location of the gas slip line. We observe a significant accumulation of the particles that have been moved along the slip line by the shear flow. The larger concentration of particles in the vicinity of triple point is, in fact, the remainder of the particles that have concentrated first at the leading edge and then were swept up with the flow. It is also interesting to observe that an essentially particle- free zone is formed due to the effects of particles slipping over the top of the cylinder and the rarefaction wave behind the cylinder.

3. CONCLUSIONS

In this paper, a computer program for two-phase compressible flow computation on adaptive grids using upwind schemes is described. The following validation study and conclusion can be made.

(1) The validation study for a one-dimensional shock wave propagating in a dusty gas shows a good agreement between the prediction of our model and the results of the experiment.

(2) For a two-dimensional gas-only flow, numerical results agree well with existing experimental data qualitatively and quantitatively, indicating that the gas phase is accurately simulated by adaptive grid technique.

(3) Particles in the gas can have a profound effect on the shock wave reflection and diffraction pattern, which is a function of particle size and loading. The smaller the particle and the lesser the particle loading, the less the inference of particle on the flow field.

(4) There is a particle accumulation behind the "back shoulder" of the semicircular cylinder due to the effect of particles inertia and gas rarefaction wave.

4. ACKNOWLEDGMENTS

The work reported here was partially supported by DARPA and AFOSR under contract no. F49620-89-C-0087. The authors would like to thank Col. J. Crowley and Dr. A. Nachman for their interest in this project. The authors also acknowledge support from DNA under contract no. DNA001-91-C-0047, and A. Frederickson for his encouragement.

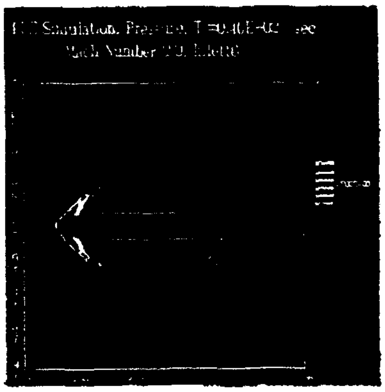
5. REFERENCES

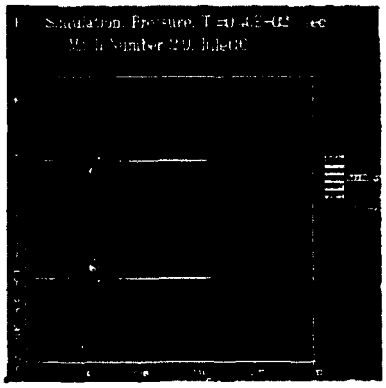
1. S. L. Soo, Particulates and Continuum, Hemisphere Publishing Corporation. 1989.
2. M. Sommerfeld, "The Unsteadiness of Shock Waves Propagating through Gas-Particle Mixtures," Experiments in Fluids, Vol. 3, p. 197, 1985.
3. R. Clift, J. R. Grace, and M. E. Weber, Bubbles, Drops and Particles, Academic Press, New York, 1978.
4. R. M. Drake, Jr., "Discussions on G.C. Vliet and G. Leppert: Forced Convection Heat Transfer from an Isothermal Sphere to Water," Journal of Heat Transfer, Vol. 83, p. 170, 1961.
5. S. Eidelman, P. Collela, and R. P. Shreeve, "Application of the Godunov Method and Its Second Order Extension to Cascade Flow Modelling," AIAA Journal, Vol. 22, p. 10, 1984.
6. P. Collela, "A Direct Eulerian MUSCL Scheme for Gas Dynamics," SIAM J. Stat. Comput., Vol. 6, p. 104, 1985.
7. P. Collela, and H. M. Glaz, "Efficient Solution Algorithms for the Riemann Problem for Real Gases," J. Comput. Physics, Vol. 59, p. 264, 1985.
8. H. Hornung, "Regular and Mach Reflection of Shock Waves," Ann. Rev. Fluid Mech., Vol. 18, p. 33, 1986.
9. J. Y. Yang, Y. Liu, and H. Lomax, "Computation of Shock Wave Reflection by Circular Cylinder," AIAA Journal, Vol. 25, p. 683, 1987.
10. J. Kaca, "An Interferometric Investigation of Diffraction of a Planar Shock Wave over a Semicircular Cylinder," UTIAS Technical Note 269, 1988.

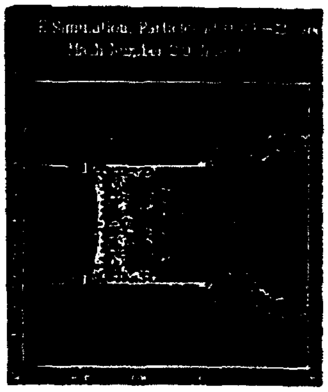
AIAA 92-3168

PULSED DETONATION ENGINE EXPERIMENTAL AND THEORETICAL REVIEW

S. Eidelman and W. Grossmann
Science Applications International Corporation
McLean, VA







AIAA/SAE/ASME/ASEE
28th Joint Propulsion
Conference and Exhibit
July 6-8, 1992 / Nashville, TN

PULSED DETONATION ENGINE
EXPERIMENTAL AND THEORETICAL REVIEW

Shmuel Eidelman and William Grossmann

Applied Physics Operation
Science Applications International Corporation
1710 Goodridge Drive, McLean, VA 22102

Abstract

A Review of past and current research on pulsed detonation engine devices connects early experimental work originating with the V1 pulsejet to recent interest in such propulsion devices. The recent interest has been, in part, stimulated by Aviation Week where sightings of aircraft contrails lead to question if some sort of PDE device has already been developed. This review summarizes what is known about PDEs, makes predictions for applications to realistic flight vehicles including missiles and full scale aircraft, and outlines what is yet required for successful PDE development.

1. Introduction

This paper reviews past and recent theoretical and experimental work related to the Pulsed Detonation Engine (PDE) concept. Such a review is timely since much interest in the PDE concept has been generated from several recent Aviation Week (AW) articles.[1,2] The AW articles, in addition to describing SAIC PDE studies, describe observations of aircraft flight and engine sound generation that are similar to what would be expected from PDE operation. These observations are intriguing since, to our knowledge, there has been no previously reported use of PDE devices in any past or recent flight vehicles. The reported observations include loud pulsing sounds at Beale AFB and photographs of high altitude contrails with "cotton ball" like beads strung on the contrails in a repetitive pattern. It is tempting to try to connect the AW reports with what we understand about PDE operation. It has come to our attention that a ground observer has identified the frequency of the pulsing sounds emanating [illegible] the vehicle that made the contrails appearing in [illegible] AW article to be of the order of 50-60 Hertz. To obtain the source (aircraft engine) frequency we must correct the observed frequency for the Doppler effect, taking into account the temperature variation between ground and flight altitude. Assuming an altitude of 45-50,000 ft. (cirrus clouds are observed behind the trails in the published photographs), a flight Mach number between one and two gives a source frequency, f, between 100 and 200 Hertz. As we show later in this paper, a PDE generating 25,000-50,000 lbs. thrust should, theoretically, operate in this frequency range. Of course we have no information concerning the subject aircraft characteristics and consequently we cannot conclude that PDEs are powering present day aircraft. On the other hand the observations appear to be consistent with expected PDE operation.

2. Early Pulsed Combustion Propulsion Devices

It is instructive to point out the differences between the PDE concept and the more commonly understood pulsejet devices. The first full scale application of pulsed propulsion devices was for the V1 flying "buzz bomb." The engine used for this vehicle was the Schmidt-Argus[3] engine and has since been generally referred to as a "pulsejet." The pulsejet for the V1 engine was based on repetitive combustion ignitions accomplished through the use of mechanical reed valves that allowed fresh air charge to be drawn into the combustion chamber. The timing of the reed valve opening was pegged to the acoustical frequency (organ pipe modes) of the combustion chamber, which consisted of a central ignition region joined to an exhaust duct. Thus, the operating parameters of the engine were fixed with engine size; only narrow ranges of thrust level variation are possible in such an engine. An increase or decrease of thrust can only be made through changes in engine internal geometry. Ever since the first occurrence of pulsejets, these engines have been considered for other applications including full scale aircraft propulsion. One of the major obstacles in the early development of the pulsejet for wider applications was the complete absence of a theoretical approach to understanding the thermodynamic process in the combustion chamber. It was assumed that the pulsejet combustion process was similar to the steady-state Lenoir constant-volume cycle and that the frequency of the combustion pulsations could be predicted by means of steady-state acoustical wave motion. However, the efficiency of the pulsejet, as determined experimentally, was much lower than a

constant-volume process would predict. We know now that the early pulsejet devices operated on an approximately constant-pressure cycle, which is known to have a lower thermodynamic efficiency than the constant-volume cycle. We have previously argued that the lack of a firm theoretical understanding of the physics and thermodynamics was primarily responsible for the failure to develop the pulsejet further for a wider range of practical applications. This argument will be discussed again later in this review.

In the meantime, the term pulsejet has become generally understood to refer to a pseudo-generic series of engines. The term "propulsive duct" is a more comprehensive descriptor encompassing a wider range of pulsed combustion engine concepts. An early series of papers by Tharjatt[4-6] described the status of work on such devices up to 1965, and provides a guide to the early attempts to understand the physics and aerodynamics of the internal gas flows in them. Even though these early investigations were seriously handicapped by a lack of knowledge of unsteady aerodynamics and the physics of repetitive combustion, it is remarkable that the conclusions offered in Tharjatt's papers are close to what we have concluded over the past several years for the PDE concept. Specifically, it was concluded that the propulsive duct engine concept should theoretically be capable of any desired level of thrust per unit area, with a corresponding reduction in specific fuel consumption. Valveless operation was also investigated and shown to offer a route to eliminating the dependency on fixed acoustical frequencies tied to a given chamber geometry. Figure 1 is representative of the valveless propulsive duct conceptualized by Tharjatt. Further, it was shown that the use of feedback techniques via multiple tube arrangements, which may not be practical from an engineering standpoint, leads to the possibility of very high frequency operation beyond the audible range. This would result in near silent operation. Finally, it was concluded that the propulsive duct should be capable of supersonic operation, and a Mach 3 engine was conceptualized; a schematic of this supersonic concept is shown in Figure 2.

Somewhat later, in a 1982 report by Kentfield,[7] the pulsejet was analyzed for predicted flight performances based on well established experimental test-stand data and available theoretical studies.[8] The results were compared against other engine alternatives suitable for small, high subsonic speed flight vehicles. The predicted performance for valveless engine configurations was shown to be highly competitive with turbojets at high subsonic Mach numbers. Actual flight tests with a drone type aircraft at Mach 0.85 showed increased performance over predicted performance values due possibly to a combination of increased air-breathing, increased intake density, and a ram effect superimposed on the pulsejet cycle. Conclusions from these studies include suggestions that valveless pulsejet performance could be comparable and, in some cases, exceed that of turbojet engines. A strong point was made concerning the low cost, simplicity and relatively high thrust-to-weight ratio of pulsejets when compared with turbojets.

The main reason for including the preceding review of pulsejets and propulsive ducts is to draw attention to the similarity between the early conclusions concerning the future performance expectations of pulsejets and the conclusions drawn to date concerning expected PDE performance. As mentioned above, we believe a primary reason that such devices have not been pursued in the past is that adequate analysis and evaluation tools did not exist at the time to help understand the complexities of pulsed operation. Modern CFD techniques now allow a comprehensive analysis of the internal and external flows associated with pulsed propulsion devices. It may well be more than just an interesting exercise to re-examine the pulsejet engines using present day CFD tools, and to compare the results with those from similar PDE studies.

3. Constant Volume Combustion and Early Pulsed Detonation Studies

Constant Volume Combustion

A constant volume combustion process is known to have a higher thermodynamic efficiency than a constant pressure combustion process. Constant volume combustion was adopted very early for use in gas turbine engine development, and the first gas turbine engines in commercial use were based on the constant volume cycle. Jet propulsion engines were one of the applications of the constant volume cycle (or explosion cycle), which was explored in the late 1940s.[8] Although the explosion cycle operates at a larger pressure variation in the combustion chamber than in a pulsejet, the cycle actually realized in these engines was not a fully constant volume one since the combustion chamber was open ended.[9] In Reference 8 the maximum pressure ratio measured in an explosion cycle engine was 3:1, whereas the pressure ratio for the same mixture under the

assumption of a constant volume cycle would be 8:1. Also, this early engine was limited by the available cycle frequency, which in turn is limited by the reaction rate. A simple calculation[8] showed that if the combustion time could be reduced in this engine from 0.006 sec to 0.003 sec, the thrust per pound of fuel-air mixture would increase 100%. Thus, a propulsion device based on an explosion-cycle has two main disadvantages:

- Constrained volume combustion (as distinguished from constant volume combustion) does not take full advantage of the pressure rise characteristic of the constant volume combustion process.
- The frequency of the explosion cycle is limited by the reaction rate, which is only slightly higher than the deflagrative combustion rate.

The main advantage of the constant pressure cycle is that it leads to engine configurations with steady state processes of fuel and oxidizer injection, combustion, and expansion of the combustion products. These stages can be easily identified and the engine designer can optimize them on the basis of relatively simple steady state considerations.

Pulsed Detonation Studies

There have been numerous attempts in the past to take advantage of detonative combustion for engine applications. The following is a brief description of some of the most relevant past experimental and analytical studies of pulsed detonation.

The Work of N. Hoffmann. The first reported work on intermittent detonation is attributed to Hoffmann[10] in 1940. Hoffmann's experiments on intermittent detonation were carried out in a long, narrow tube mounted on a test stand using acetylene-oxygen and benzine-oxygen fuel mixtures. Water vapor was added to prevent the highly sensitive acetylene-oxygen mixture from premature detonation. Hoffmann pointed out the importance of the detonation initiation (spark plug) location in reference to tube length and diffuser length. It was found that a continuous injection of the combustible mixture leads to only a narrow range of ignition frequencies that will produce an intermittent detonation cycle. These frequencies are governed by the time required for the mixture to reach the igniter, time of transition from deflagration to detonation, and time of expansion of the detonation products. Hoffmann attempted to find the optimum cycle frequency experimentally. It was discovered that detonation-tube firing occurred at lower frequencies than the spark-plug energizing frequencies, indicating that the injection flow rate and ignition were out of phase. Wartime events prevented further work by Hoffmann and his co-workers.

The Work of Nicholls and Co-Workers. A substantial effort in intermittent detonation research was made by a group headed by J. A. Nicholls[11-12] of the University of Michigan beginning in the early 50's. The most relevant work concerns a set of experiments carried out in a six foot long detonation tube.[11] The detonation tube was constructed from a one inch internal diameter stainless steel tube. The fuel and oxidizer were injected under pressure from the (closed) left end of the tube and ignited at some distance down stream. The tube was mounted on a pendulum platform, suspended by support wires. Thrust for single detonations was measured by detecting tube (platform) movement relative to a stationary pointer. For multi-cycle detonations, thrust measurement was achieved by mounting the thrust end of the tube to the free end of a cantilever beam. In addition to direct thrust measurements, the temperature on the inner wall of the detonation tube was measured. Fuel mixtures of hydrogen/oxygen, hydrogen/air, acetylene/oxygen and acetylene/air mixtures were used. The gaseous oxidizer and fuel were continuously injected at the closed end of the detonation tube and three fixed flow rates were investigated. Under these conditions, the only parameters that could be varied were the fuel/oxidizer ratio and frequency of ignition. A maximum gross thrust of ~ 3.2lb was measured in the hydrogen/air mixture at the frequency of ~ 30 detonations per second. The most promising results were demonstrated for the H_2/air mixture, where a fuel specific impulse of I_{sp} = 2100 sec was reached. The maximum frequency of detonations obtained in all experiments was 35 Hz. The temperature measurements on the inner wall showed that for the highest frequency of detonations the temperature did not exceed 800° F. This temperature is approximately the mean between the temperature of the injected gasses and the detonation wave temperature averaged over the cycle frequency.

In their later work,[13-15] the University of Michigan group concentrated on development of the Rotating Detonation Wave Rocket Motor. No further work on the pulsed detonation cycle was pursued.

The Work of L. J. Krzycki. In a setup very similar to Nicholl's, L. J. Krzycki[16] performed an

experimental investigation of intermittent detonations with frequencies up to 60 cps. An attempt was also made to analyze the basic phenomena using unsteady gas dynamic theory. Krzycki's attempt to analyze the basic phenomena relied on wave diagrams to trace characteristics, assumptions of isentropic flow for detonation and expansion, and incompressible flow for mixture injection processes. The most convincing data from the experiments are the measurement of thrust for a range of initiation frequencies and fuel mixture flow rates. Unfortunately no direct pressure measurement in the device is reported, so there is only indirect evidence of the nature of the process observed.

The basic test stand used by Krzycki is very similar to that used by Nicholls and his co-workers. The length of the detonation tube and the internal diameter were exactly the same as those in Nicholl's experiments. Figure 3 presents a schematic of the experimental apparatus containing common, generic elements of the Hoffmann-Nicholls-Krzycki experiments. A propane/air mixture was continuously injected through a reversed-flow diffuser for better mixing, and was ignited at the same distance as in the Nicholls' experiments from the injection point by an automobile spark plug. The spark frequency was varied from 1 to 60 cps. The spark plug power output was varied inversely with the initiation frequency, and at the frequency of 60 cps was only 0.65 Joule. This value is too low for direct initiation of a detonation wave by the spark, and consequently all of the experiments must have been based on transition from deflagration to detonation. According to experimental data and theory,[17] direct initiation of a mixture of propane/air at the detonability limits requires an energy release on the order of 10^6 Joules. Thus, we conclude that the required deflagration-detonation transition region length in Kryzcki's experiments would have been prohibitively large for the propane/air mixture. It follows that in all of the experiments a substantial part of the process was deflagrative. This resulted in low efficiency and negligible thrust. Krzycki repeated Nicholls' experiments using basically the same rates of injection of the detonable mixtures. Krzycki's experimental results are very well documented, allowing us to deduce a clear picture the physical processes occurring in the tube. The author arrived at the conclusion that thrust was possible from such a device but practical applications did not appear promising. It is unfortunate that, possibly based on Krzycki's extensive but misleading results, all experimental work related to the pulsed detonation engine concept stopped at this time.

Russian Work on Pulse Detonation Devices. A review of the Russian literature has not uncovered work concerning applications of pulsed detonation devices to propulsion. However, there are numerous reports of applications of such devices for other purposes such as for producing nitrogen oxide[18] (an old Zeldovich idea to bind nitrogen directly from air to produce fertilizers) and as rock crushing devices.[19]

Korovin et al.[18] provide a most interesting account of the operation of a commercial detonation reactor. The main objective of this study was to examine the efficiency of thermal oxidation of nitrogen in an intermittent detonative process as well as an assessment of such technological issues as the fatigue of the reactor parts exposed to the intermittent detonation waves over a prolonged time. The reactor consisted of a tube with an inner diameter of 16 mm and length 1.3 m joined by a conical diffuser to a second tube with an inner diameter of 70 mm and length 3 m. The entire detonation reactor was submerged in running water. The detonation mixture was introduced at the end wall of the small tube. CH_4, O_2 and N_2 comprised the mixture composition and the mixture ratios were varied during the continuous operation of the reactor. The detonation wave velocity was measured directly by piezoelectric sensors placed in the small and large tubes. The detonation initiation frequency in the reactor was 2-16 Hz. It is reported that the apparatus operated without significant maintenance for 2000 hours.

Smirnov and Boichenko[19] studied intermittent detonations of gasoline-air mixtures in a 3 m long and 22 mm inner diameter tube operating in the 6-8 Hz ignition frequency range. The main motivation for this work was to improve the efficiency of a commercial rock crushing apparatus based on intermittent detonations of the gasoline/air mixtures.[20] The authors investigated the dependence of the transitional region length from deflagration to detonation on the initial temperature of the mixture.

As a result of the information contained in the Russian reports, we conclude that reliable commercial devices based on intermittent detonations have been constructed and operated.

Pulsed Solid Explosion Studies at JPL. Work at the Jet Propulsion Laboratory (JPL) by Back, Varsi and others[21-24] concerned an experimental and

theoretical study of the feasibility of a rocket thruster based on intermittent detonations of solid explosive for propulsion in dense or high-pressure atmospheres of certain solar system planets. The JPL work was directed at very specific applications: however, these studies also addressed more general key issues concerning intermittent propulsion devices such as propulsion efficiency. In this work, a Deta sheet type C explosive was detonated inside a small detonation chamber attached to nozzles of various length and geometry. The nozzles, complete with firing plug, were mounted in a containment vessel that could be pressurized with mixtures of various inert gases from vacuum to 70 atm. The apparatus directly measured the thrust generated by single detonations of a small amount of solid explosive charge expanding into conical or straight nozzles. Thrust and specific impulse were measured by a pendulum balance system.

The results obtained from the JPL experimental study of an explosively driven rocket led to the following conclusions. First, rockets with long nozzles show increasing specific impulse with increasing ambient pressure in CO_2 and N_2. Short nozzles, on the other hand, show that specific impulse is independent of ambient pressure. Most importantly, most of the experiments obtained a relatively high specific impulse of 250 seconds and larger. This result is all the more striking since the detonation of a solid explosive yields a relatively low energy release of approximately 1000 cal/gm compared with 3000 cal/gm obtained in hydrogen oxygen combustion. Thus, it can be concluded that the total losses in a thruster based on unsteady expansion are not prohibitive and hence, in principle, very efficient intermittent detonation propulsion systems are possible.

4. Description of the PDE Concept

Basic Principles

A detonation process, due to the very high chemical reaction rate in the detonation wave, leads to a propulsion concept in which the constant volume process can be fully realized. In detonative combustion, a strong shock wave, which is part of the detonation wave, acts like a valve between the detonation products and fresh charge; the detonation wave functions at the same time as a valveless compressor between the fresh fuel/air mixture and the detonation products. The speed of the detonation wave is about two orders of magnitude higher than the speed of a typical deflagration wave. Because of this, very high power densities can be created in the detonation chamb... Each detonation can be initiated independently and, depending on the chamber geometry and external flow characteristics pertaining to a particular device, a wide range of frequencies is possible. There is no theoretical restriction on the range of operating frequencies: they are uncoupled from any acoustical chamber resonance. The independence of detonation cycle frequency is the feature that most differentiates the PDE concept from the pulsejet. It is also the feature that leads theoretically to scalability of PDE configurations for a wide range of flight applications. A key physical restriction on the range of allowable detonation frequencies arises from the rate at which the fresh fuel/air mixture can be introduced into the detonation chamber. Obviously the detonation products must be discharged from the chamber before fresh charge is injected.

First PDE Experiments

To our knowledge, the first experiments that successfully demonstrated repetitive or pulsed detonation was attainable in a propulsion-like device were carried out by Helman, Shreeve and Eidelman[25] at the Naval Postgraduate School in 1985-86. During these studies, several fundamentally new ideas were developed for pulsed detonation applications to propulsion. First, to overcome the energy requirements for detonation initiation, a pre-detonation was initiated in a small detonation tube where an oxygen rich fuel mixture could be detonated at substantially lower energies than those required for full fuel/air mixtures. Next, the experimental PDE was operated in a self-aspirating mode; the detonation exhaust gases were discharged through gasdynamic expansion and fresh air was drawn into the detonation chamber due to chamber overexpansion following detonation product exhaust. Figure 4 is a schematic of one of the variations of the PDE experimental configurations. The pre-detonation initiation tube is shown attached to a spark plug. The most important results were obtained when the fuel injection (injection was accomplished with a toroidal ring containing holes near the exhaust plane of the device) rate was timed appropriately (the lag time between the fuel/air travel to the pre-detonation port and the arrival of the pre-detonation pulse) with detonation initiation. The principle of repetitive detonation initiation and control was definitively established in these experiments. Pressure transducer traces unambigiously showed that a detonation wave was

formed in the chamber and propagated with the Mach number appropriate for the fuel-air mixture. The fuel used in the NPS experiments was ethylene and the maximum detonation frequency obtained was 25 Hz, limited only by the mechanical nature of the solenoid valve used for fuel injection control. Figures 5 and 6 are two frames from a videotape of the early NPS experiments. Figure 5 shows the experimental apparatus and Figure 6 shows the apparatus during repetitive detonation. The figures also show the fuel injector ring between the two concentric detonation chamber cylinders. It was determined that the duration of a single cycle was less than 7 msec. This means that the NPS device could have potentially operated at frequencies up to 150 Hz in the static or no flow (M = 0) case. At the time of the NPS experiments, performance extrapolations included thrust levels up to 40 lbs at 100 Hz. As described later, SAIC simulations of static operation show higher thrust levels at these frequencies due to new ideas and improvements in the PDE concept. These new ideas are incorporated in the generic PDE concept.

The Generic PDE Device

In this section, we refer to the generic PDE device, which is represented as a small engine in Figure 7. The figure shows a schematic of the basic detonation chamber attached to the aft end of a generic aerodynamic vehicle. A combustible gas mixture is injected at the closed end of the detonation chamber and a detonation wave is shown propagating through the mixture. Also shown are air injection inlets and an important part of the device that we have termed the thrust wall. The schematic suggests a small-payload aerodynamic vehicle; however, as we describe later, the concept can be extended to larger payloads simply by scaling up the size of the detonation chamber and possibly combining a number of chambers into one larger engine.

The geometry of the main detonation chamber, which determines the propulsion efficiency and the duration of the cycle (frequency of detonations), is a key issue for the PDE concept. Since the fresh charge for the generic engine is supplied from the external flow field, the efficiency of the engine depends on the interaction of the surrounding flow with the internal flow dynamics. Following is a partial list of the broad range of physical processes requiring simulation in order to model the complex flow phenomena associated with the detonation engine performance:

1. Initiation and propagation of the detonation wave inside the chamber;

2. Expansion of the detonation products from the chamber into the air stream around the chamber at flight Mach numbers;

3. Fresh air intake from the surrounding air into the chamber;

4. The flow pattern inside the chamber during post-exhaust pressure buildup, which determines the strategy for mixing the next detonation charge;

5. Strong mutual interaction between the flow inside the chamber and the external flow surrounding the engine.

All of these processes are interdependent, and interaction and timing are crucial to engine efficiency. Thus, unlike simulations of steady state engines, the phenomena described above cannot be evaluated independently. It is a challenging computational problem to resolve the flow regime inside the chamber to account for nozzles, air inlets, etc., and at the same time resolve the flow outside and surrounding the engine, where the flow regime varies from high subsonic, locally transonic and supersonic.

The single most important issue is to determine the timing of the air intake for the fresh charge that leads to repetitive detonations. It is sufficient to assume inviscid flow for the purpose of simulating the expansion of the detonation products and fresh air intake. The assumption of inviscid flow makes the task of numerically simulating the PDE flow phenomena somewhat easier than if a fully viscous flow model were employed. The effects of viscous boundary layers are negligible for the size of the generic device studied in this work, with the exception of possible boundary layer effects on the valve and inlet geometries discussed subsequently.

SAIC has performed an extensive study of the generic PDE over a wide range of operating conditions for a wide range of device configurations.[26-30] Numerical simulations of the unsteady flow and detonation processes, in addition to theoretical analysis, have resulted in an understanding and an approach to analyzing and evaluating PDE propulsion performance. Although the basic concept remains the same, there are subtle differences in the PDE manifestation for particular applications. These will be described subsequently. Details of the

numerical simulations (including assumptions used for detonation wave physics and chemistry, use of adaptive unstructured grids and Godunov methods for the Euler gasdynamic equations) are given elsewhere.[26-31] The following section is a summary of the results from numerical and theoretical studies of various applications and operating regimes for the generic PDE.

5. Operating Regimes

In this section we summarize the results of several applications and operating regimes identified in the course of our studies of the PDE concept.

M = 0 Static Operation

Under static conditions, M = 0, the PDE is completely self-aspirating. Such was the case for the early NPS PDE studies. Without an external airstream, the PDE must obtain fresh air charge as a result of the detonation chamber overexpansion immediately following exhaust of air-fuel detonation products. To the lowest approximation, the available time for chamber refill due to this overexpansion process is, for a given chamber geometry and fuel-air combination, directly proportional to its length. For M = 0 operation, we assume that the PDE configuration does not contain any air inlets other than the aft end of the device or, if inlets are present, they are closed. Simulations[26] of M = 0 PDE operation show that the time required for fresh air refill for a device with dimensions equivalent to the NPS experimental apparatus is on the order of 6-7 msec. This agrees with the NPS results and means that a maximum frequency of 150 Hz should be possible. Simulated thrust levels were higher than those estimated from scaling the NPS results. This is due to a new operating scenario that was uncovered by the simulations: detonation initiation from the aft end results in the kinetic energy of the shock wave being transferred to the thrust wall. The amount of extra thrust obtained from this mode of operation is considerably larger than that expected from gasdynamic expansion following detonation initiation at the thrust wall. The physical reason for this is found in the shock wave energetics.

The importance of M = 0 PDE performance is associated with applications of the concept for full scale aircraft propulsion, including rollout and takeoff. Simple scaling laws derived from the numerical simulation results and described later, show that M = 0 thrust levels can be large (tens of thousands of lbs.) depending on the engine cross sectional area, length and detonation frequency.

Subsonic-Transonic Operation

PDE operation in the subsonic-transonic regime differs from the static case in that the self aspiration effect decreases with increasing Mach number. This is due to the formation of a rear stagnation point behind the exhaust plane above certain Mach numbers for given geometries. The stagnation region prevents complete detonation product exhaust and subsequent fresh charge injection. For example, over the Mach number range, $0^+ < M < 0.5$, full to partial self aspiration occurs; the effect decreases rapidly for Mach numbers above 0.5, resulting in the need for some type of air inlet or air intake valve configuration. Simulations of various detonation chamber and air inlet geometries[26,28] have shown that, depending on the free-stream Mach number, appropriate shaping of the air inlet geometry and total inlet area leads to propulsion engines that are attractive for certain applications. We present here a summary of studies[28] carried out in an attempt to find a satisfactory PDE configuration for a small missile engine (the final configuration was not optimum, by any means, since all variables were not parametrically varied).

A PENAID-type missile with associated mission requirements such as range, speed, system weight, total thrust, and specific fuel consumption was used for the study. The detonation chamber dimensions were 6 cm diameter and 9 cm length with a cylindrical cross-section. A schematic of PDE integration into such a missile configuration is shown in Figure 8. The simulations showed that, for practically all cases involving simple inlets (circumferential slits around the cylindrical cross-section), the thrust data were independent of whether the inlets open intermittently (valved) or remain open during operation. This is due partially to the very short time that detonation products have to escape from the inlets thereby adding to negative thrust; this negative thrust, determined in the simulations, is negligible compared to the total integrated thrust. The thrust data do indicate a strong dependence on external flow conditions, e.g., Mach number. The Mach number plays a role in the wave drag; the details of valve and inlet configuration geometry figure prominently in the total wave drag. These studies answered an important question: can an air inlet be configured such that the inlet remains open over the full flight regime and operating conditions? The answer is "yes." Thus, at least for this regime, the PDE offers the possibility of a no-

moving-parts propulsion device. For the PENAID missile under discussion here, a configuration was found that operates between $0.2 < M < 0.9$ with open air inlets.

The following performance data were obtained for the PENAID missile configuration. For $M=0.8$ at sea level altitude and a detonation frequency, $f=100$ Hz, the PDE characteristics are:

Thrust	79 lb.
Fuel flow rate	0.025 lb./sec.
Fuel weight for 12 min	18 lb.
Oxygen weight	1.8 lb
Fuel for detonation tube	0.6 lb.
Total oxygen and fuel weight	20.4 lb.
Total engine weight	30.2 lb.
Specific fuel consumption	1.14 lb./(lb.*hr.)

Assuming the PDE device geometry is kept fixed, a higher detonation frequency will result in a linear increase in thrust and fuel flow rate at the same specific fuel consumption. For example, if the detonation frequency is increased to 200 Hz, the performance data are:

Thrust	157 lb.
Fuel flow rate	0.05 lb/sec.
Fuel weight for 12 min	36 lb.
Oxygen weight	3.6 lb.
Fuel for detonation tube	1.2 lb.
Total oxygen and fuel weight	40.8 lb.
Total engine weight	54.4 lb.
Specific fuel consumption	1.14 lb./(lb.*hr.)

At lower Mach numbers, $M=0.5$, the maximum operating frequencies for constant thrust will be lower since the external dynamic pressure responsible for supplying fresh air to the chamber is also lower. For the device under consideration here, the maximum frequency is 250 Hz. For a frequency of 100 Hz:

Thrust	100 lb.
Fuel flow rate	0.025 lb/sec.
Fuel weight for 12 min	18 lb.
Oxygen weight	1.8 lb.
Fuel for detonation tube	0.6 lb.
Total oxygen and fuel weight	20.4 lb.
Total engine weight	30.2 lb.
Specific fuel consumption	0.9 lb./(lb.*hr.)

Again, if the frequency is increased the thrust will increase linearly; operation at 200 Hz yields:

Thrust	200 lb.
Fuel flow rate	0.05 lb/sec.
Fuel weight for 12 min	36 lb.
Oxygen weight	3.6 lb.
Fuel for detonation tube	1.2 lb.
Total oxygen and fuel weight	40.8 lb.
Total engine weight	54.2 lb.
Specific fuel consumption	10.9 lb./(lb.*hr.)

The examples of the PDE device performance given above are based on point design conditions arising from the simulations reported earlier.[26] They cannot be extrapolated with any degree of reliability to other conditions or configurations. We conclude, however, that the performance computed for the indicated device is encouraging from the point of view of thrust, thrust control, simplicity of the device (no moving parts), and specific fuel consumption (SFC). The specific fuel consumption computed above is competitive with present day small turbojet engines. The SFC for a PDE could be significantly lower than for small turbojets (SFC's for small turbojets are in the range of 1.8-2.0 lb./(lb.*hr)). Thus, for a given mission and vehicle, a PDE propulsion unit may be more fuel efficient, resulting in increased range. Moreover, if the expected thrust control in PDE's is realizable, it may be possible to produce propulsion units that can slow down, loiter and maneuver, and finally accelerate to full thrust again rapidly. Depending on the detonation frequency, which determines the thrust for all other conditions fixed, the thrust-to-weight ratio for the PDE can be as high as 20:1. This value is certainly competitive with other propulsion concepts.

The results of the scaling studies at subsonic-transonic speeds lead to scaling laws that can be used to predict the performance of PDE's over some range of parameters, assuming that other parameters are held fixed. For example, holding the external Mach number and basic chamber and inlet geometry fixed suggests that the thrust at constant specific fuel consumption produced by the PDE scales as:

$$\text{Thrust} = T_1 * \left(\frac{v}{v_1}\right) * \left(\frac{f}{f_1}\right),$$

where T_1, (v/v_1) and (f/f_1) are the thrust computed for a chamber of volume v_1 operating at frequency f_1, the ratio of a new volume to v_1 and the ratio of the new frequency to f_1, respectively. Thus, thrust should scale linearly with the parameter $(v/v_1) * (f/f_1)$ over some range of this parameter. Departure from this linear variation may occur due to the following argument: First, since volume is proportional to the

product of cross-sectional area and length, $v \sim r^2 l$, (r ~ detonation chamber radius, l ~ chamber length) physical limits will be placed on r and l: if r is too small (less than 1 cm), a detonation will not be sustainable and if l is too small (less than 10 cm), it may be difficult to mix fuel and air effectively. Using the thrust relation established above, we make the following observations. For a PDE device producing 100 pounds thrust at 100 Hz, doubling the frequency and increasing the volume by a factor of 5 yields a thrust level of 1000 pounds. Assuming that the aspect ratio of the chamber (chamber length to radius) is fixed, this would require an engine only 25.5 cm in diameter and 25.5 cm in length. Of course, the relation between thrust and $(v/v_1) * (f/f_1)$ cannot be believed over too wide a range of parameters; but, it does serve to point out the flexibility permitted by the PDE concept.

The subsonic-transonic simulations showed that the timing of the fresh air refilling required to recharge the chamber for subsequent detonations is a strong function of the details of the valve and inlet geometry, the expansion of the combustion products, the resulting over-expansion of the chamber flow, and the external flow regime and interaction of the external flow with the internal flow. For subsonic flight, Mach 0.2-0.9, the fresh air entering the chamber comes from two separate principal flow processes; one comes from the flow through any valve or inlet and the other comes from the self-aspiration or reverse flow from the aft end of the chamber due to strong over-expansion. All these processes are interdependent and, in order to search for a given performance in a given device, require variation of many parameters. The simulation results obtained to date provide an understanding of the effects caused by variation of the above-mentioned parameters. With the information available, we conclude that a PDE propulsion unit can be optimized (although no optimization studies were carried out) for a given flight regime. The decrease in thrust with increasing Mach number has been described earlier to result from increased wave drag produced by the inlet geometry. Optimization of the inlet geometry could help to eliminate a large part of the wave drag. The simulation data can be used to determine the detonation frequency at a given Mach number yielding constant thrust. For example, for a constant thrust level of 90 pounds, the required detonation frequency varies from 84 Hz at M-0.0 to 140 Hz to M=0.8. In a similar fashion, we can obtain parametric variations of other important aspects of PDE performance, such as minimum time for refill at given Mach number as a function of air inlet opening. To find an optimum configuration that satisfies given performance over a wide flight regime requires a more extensive simulation study. It was mentioned earlier that the simulations presented here were carried out under the assumption of inviscid flow: boundary layer effects were not included. Boundary layers are only significant for the air inlets and valves.

There is an important feature of PDE operation for missiles such as the one considered here: if the expected thrust control is attainable, then the detonation frequency can be varied to produce constant thrust over a given flight envelope, or the frequency can be varied to make the missile slow down, loiter and maneuver, and finally ramp back to full thrust more or less instantaneously. Since each detonation is controlled separately, this capability should depend only on on-board electronics and power.

Supersonic-Hypersonic Operation

Numerical simulations have been carried out for PDE operation in the supersonic and hypersonic flight regimes.[29] The results of these simulations show that there are differences when compared with the lower speed regimes. The main difference, with respect to operating characteristics, is the air intake inlet must be more carefully considered. For supersonic and hypersonic flow air scoops may be required, adding to wave drag. For PDEs enclosed in a duct connected to upstream air inlets, pressure recovery from free-stream to duct inlet and finaly to PDE inlet must be accounted for. To date, several detailed studies have been carried out for the higher speed regimes; a supersonic, M = 2 PENAID missile engine simulation and a sizing analysis for a large engine operating in the supersonic to hypersonic flight regime.

Supersonic M = 2 PDE The M = 2 PENAID missile study has been reported earlier[29] and, representative simulation results are shown on the cover of this review paper. It was found that a fixed air inlet geometry could be conceptualized to operate over the Mach number range, $0.5 < M < 2$. By this is meant the timing for fresh air charge allowed a detonation frequency of 200 Hz at M = 2 and this, in turn, means that any lower frequency is allowable at any other Mach number below M = 2. Detonation frequency control may result in enhanced control over missile flight trajectory since a constant thrust, a cruise-dash-loiter-cruise or any other tailored thrust profile can be realized. We conclude that supersonic PDE operation appears possible for missile

applications. and there may also be advantages for longer range air-to-air missiles due to enhanced propulsion energy management capability.

Sizing Analysis for Large PDEs A zeroth order sizing analysis has been carried out to define and size a PDE configuration satisfying high thrust level requirements from sea level to 30,000 ft altitude and for a flight trajectory including the Mach number range, $0 < M < 4$. The nominal target thrust level was 50,000 pounds and we assume that the aircraft/engine integration requires an air inlet duct to deliver fresh air to the PDE. We sketch here an outline of the analysis and give the main results.

We use the simple scaling argument given and use the thrust data obtained from simulations of the smaller missile configurations. We also assume a nominal detonation frequency, $f = 100$ Hz. We then establish the following baseline PDE performance operating point. At 3×10^4 ft. altitude for $M = 2$ the thrust in pounds per cubic meter detonation chamber volume is 2.5×10^4 lbs/m^3. Therefore, an engine producing 5×10^4 pounds thrust requires a 2 m^3 chamber volume. The sizing study answers the following questions : what is the size and shape of the detonation chamber, required detonation chamber air inlet areas, frequency variation range, and effect of air inlet duct losses on a PDE developing the nominal target thrust?

We denote free-stream conditions by $(\ )_0$, PDE air inlet conditions by $(\ )_2$, and PDE detonation chamber conditions by $(\ )_3$. To account for air inlet duct losses we define the ratio of PDE inlet total pressure to free-stream total pressure by C or:

$$\frac{P_{t2}}{P_{t0}} = C. \qquad (1)$$

The simplest condition to assume for the PDE air inlet is choked flow. Although this is not valid over much of the required regime, certainly not for subsonic external flow, it will result in a pessimistic bound on the sizing parameters. Using well known gasdynamic analysis[32] the static and total pressures and density at the PDE inlet can be found as:

$$P_{t_2} = CP_0\left(1 + \frac{M_0^2}{5}\right)^{\frac{7}{2}} \qquad (2)$$

$$P_2 = P_0 C\left(\frac{5 + M_0^2}{6}\right)^{\frac{7}{2}} \qquad (3)$$

$$\rho_2 = 1.2\, C\, \rho_0\left(\frac{5 + M_0^2}{6}\right)^{\frac{7}{2}}\left(\frac{5 + M_0^2}{5}\right)^{-1} \quad (4)$$

The mass flow rate through the engine inlet is:

$$\dot{m} = \rho_2 U_2 A_2, \quad (5)$$

and, using equations 2-4, gives:

$$\dot{m}_2 = A_2\left(1.2\,\gamma\, C^2\left(\frac{P_0^2}{RT_0}\right)\left(\frac{5 + M_0^2}{6}\right)^7\left(\frac{5 + M_0^2}{5}\right)^{-1}\right)^{\frac{1}{2}} \quad (6)$$

An equation for the area ratio A_2/A_3 can be found as:

$$\frac{A_2}{A_3} = \frac{216}{125} M_3\left(1 + \frac{M_3^2}{5}\right)^{-3}, \qquad (7)$$

where M_2 has been set equal to unity. Our analysis does not include the thermodynamics of the PDE cycle; the sizing analysis is based totally on a determination of the allowable detonation frequencies in the PDE chamber. We obtain a bound on allowable flow speeds in the detonation chamber by requiring the detonation chamber to refill in the time between detonations. We further require the fuel to mix and flow with the mean speed U_3 from inlet to chamber exit, a distance equal to L, the chamber length. Thus, we obtain the relation $U_3 = f\,L$, where f is the detonation frequency. A calculation of M_3 gives:

$$M_3 = \frac{U_3}{U_3^*} = f\,L\sqrt{\frac{\rho_3}{\gamma P_3}}. \quad (8)$$

Since the total pressure in the chamber equals the total pressure at the PDE inlet, the static pressure in the chamber as a function of chamber Mach number, given in Eq. (8), can be related to the free-stream static pressure as follows:

$$P_3 = CP_0\left(1 + \frac{M_0^2}{5}\right)^{\frac{7}{2}}\left(1 + \left(\frac{A_2}{A_3}\right) C_1 \frac{Lf}{\gamma}\frac{1}{5P_3}\right)^{-\frac{7}{2}} \quad (9)$$

where C_1 is:

$$C_t = \left(1.2\,\gamma\, C^2 \left(\frac{P_0^2}{RT_0} \right) \left(\frac{5 + M_0^2}{6} \right)^7 \left(\frac{5 + M_0^2}{5} \right)^{-1} \right)^{\frac{1}{2}}$$

Another relation between P_3 and P_0 as a function of M_3 can be given as:

$$P_3 = CP_0 \left(1 + \frac{M_0^2}{5} \right)^{\frac{7}{2}} \left(1 + \frac{M_3^2}{5} \right)^{\frac{-7}{2}} \quad (10)$$

Equations (7), (9) and (10) form a closed set for the variables P_3, A_2/A_3 and M_3 with parameters C, P_0, M_0, L, f, T_0, g, and R, the universal gas constant. The volume, V, of the detonation chamber is given by the product, $V = L\,A_3$. Thus, for a given volume, Equations (7), (9), and (10) can be solved for A_2/A_3 versus L or A_3. Figure 9 gives a schematic of the PDE showing the air inlet gap width "l" resulting in an inlet area of A_2, the detonation chamber length L, and the chamber cross-sectional area A_3. We choose first a square chamber cross-section; the total inlet area is therefore given by the expression $A_2 = 4l\,(A_3)^{1/2}$. Results obtained from solving Eqs. (7), (9) and (10) are presented in Figure 10 for the baseline conditions. There, the area ratio, A_2/A_3, is given versus A_3. If A_3 is chosen to be 1.2 m^2 then the length of the PDE is 1.67 m and the engine inlet opening is 15 cm. Also shown in Figure 10 is the effect of C, the pressure recovery factor. The range of values chosen for C was: $0.7 < C < 1$. The effect of C is negligible for the range studied here. More realistic estimates for duct losses resulting in much lower values of C at high Mach numbers may well have a more pronounced effect. If the cross-sectional area is held fixed, Eqs. (7), (9) and (10) yield the results shown in Figure 11. The curve cannot be extended below $M = 1$ since the assumption of choked flow at A_2 is not valid; indeed, the assumption is not valid somewhere before $M = 1$ due to duct loss effects. The results from Figure 11 can be translated into inlet gap widths as shown in Figure 12. Figure 12 shows a range of inlet openings that, when compared with the total engine length, is equivalent to 8-12% of the total engine length. Below $M = 1$, a combination of self- aspiration and recharge from air inlets must be considered depending on Mach number. For self-aspiration at $M = 0$, the ratio of A_2/A_3 is unity; the inlets are not needed. For Mach numbers between zero and say, 0.5, partial air inlet opening is required and for Mach numbers greater than 0.5, the inlets will be fully open. For a fixed PDE configuration, varying the detonation frequency changes the thrust according to the scaling law given earlier. Figure 13 shows the effect of frequency variation on A_2/A_3. Recall the design point was at $f = 100$ Hz. Figures 10-13 contain the answers to the questions asked during this sizing analysis: reasonable physical sizes for PDEs developing high thrust levels are predicted. A more rigorous analysis is required to validate these predictions.

To conclude this section, we show the variation of thrust as a function of chamber volume derived from the baseline conditions used above. Figure 14 gives this variation and, if a circular cross-section engine is considered, varying the baseline thrust yields engine sizes shown in Figure 15. For example, a 45,000 pound thrust engine 1.67 meters long has an engine diameter of 1.2 meters. This number is not unreasonable and compares well with sizes of current turbojet engines. As mentioned, a more detailed analysis of PDE performance is needed, including an effective "steady state" thermodynamic cycle model, to validate the PDE as a credible alternative for high thrust propulsion engines.

6. Summary and Conclusions

Past and recent studies have shown that pulsed propulsion devices theoretically offer significant advantages over steady state engines. The advantages range from the possibility of a no-moving-parts configuration to high thermodynamic efficiency constant volume cycles. Numerical simulations, theoretical analysis and scaling studies of PDE performance have shown applicability to many different flight vehicles including small missiles and full scale aircraft. Configurational flexibility offered by the PDE include non-circular cross-sectional detonation chambers allowing consideration of unique aircraft/engine integration possibilities. Thus, the numerical simulation and theoretical studies of PDE performance to date have shown interesting and important propulsion applications.

In order to realize the PDE potential, experimental data is required to validate the theoretical predictions and, most importantly, provide a proof of principle demonstration of the PDE mode of operation described in this paper, namely, detonation initiation from the exhaust end of the engine. The principle of sustained repetitive detonation has already been demonstrated in the NPS experiments, but, this took place at the inner thrust wall. The next step in the development of practical PDE devices requires a comprehensive experimental program where such key

issues as detonation initiation, air inlet design including boundary layers, fuel/air injection and mixing can be studied and understood. In addition, thrust measurements, both static and in an external flow are required to validate the numerical and theoretical predictions. Plans for such an experimental program are presently under consideration.

Acknowledgments

The work reviewed in this paper received partial support from the Defense Advanced Research Projects Agency under the following contracts: N66001-88-D-0088 (NOSC) and F49620-89-C-0087 (AFOSR). The authors would like to acknowledge Robert Pegg and Larry Hunt of NASA LaRC, John Leingang of the Wright Aeronautical Laboratory, and Major David Neyland of DARPA for their interest and for stimulating discussions concerning PDE applications. Over the course of these studies, SAIC colleagues Ed Rogan, Adam Drobot, Isaac Lottati and Ahron Freidman provided support and advice on many of the technical areas required to carry out this work.

Note Added in Proof The authors would like to thank Mr Robert Pegg of NASA's Langley Research Center for bringing their attention to early documents and reports describing pulsed detonation engine research.

References

1. Scott, William B., "Renewed Interest in Pulsed Engines May be Linked to Black Aircraft," Aviation Week, 28 October 1991 (68-69).
2. Scott, William B., "New Evidence Bolsters Reports of Secret, High-speed Aircraft," Aviation Week, May 11, 1992 (62-63).
3. Wolfe, M.O.W., Luck, G.A., "Pressure Measurements on the F.Z.G. 76 Flying Bomb Motor," Technical Note No. EA237/1, Royal Aircraft Establishment, Farnborough, 1944.
4. Tharratt, C.E., "The Propulsive Duct," Aircraft Engineering, November 1965, (327-337).
5. Tharratt, C.E., ibid, December 1965, (359-371).
6. Tharratt, C.E., ibid, February 1966, (23-25).
7. Kentfield, J.A.C., "Valveless Pulsejets and Allied Devices for Low Thrust, Subsonic, Propulsion Applications," AGARD Conf - Proc. No. 307, Ramjets and Rockets for Military Applications, March (1982).
8. Zipkin, M.A. and Lewis G.W., "Analytical and Experimental Performance of an Explosion-Cycle Combustion Chamber of a Jet Propulsion Engine," NACA TN-1702, September 1948.
9. Shultz-Grunow, F., "Gas-Dynamic Invesugation of the Pulse-Jet Tube," NACA TM-1131, February 1947.
10. Hoffmann, N., "Reaction Propulsion by Intermittent Detonative Combustion," Minisrry of Supply, Volkenrode Translation, 1940.
11. Nicholls, J.A., Wilkinson, H.R. and Morrison, R.B., "Intermittent Detonation as a Thrust-Producing Mechanism," Jet Propulsion, 27, 534-541, 1957.
12. Dunlap, R., Brehm, R.L. and Nicholls, J.A., "A Preliminary Study of the Application of Steady State Detonative Combustion of a Reaction Engine," ARS J., 28, 451-456, 1958.
13. Nicholls, J.A., Gullen, R.E. and Ragland K.W., "Feasibility Studies of a Rotating Detonation Wave Rocket Motor," Journal of Spacecrafts and Rockets, 3, 893-898, 1966.
14. Adamson, T.C. and Olsson, G.R., "Performance Analysis of a Rotating Detonation Wave Rocket Engine," Astronautica Acta, 13, 405-415, 1967.
15. Shen, P.I. and Adamson, T.C., "Theoretical Analysis of a Rotating Two-Phase Detonation in Liquid Rocket Motors," Astronautica Acta, 17, 715-728, 1972.
16. Krzycki, L.J., Performance Characteristics of an Intermittent Detonation Device, Navweps Report 7655, U.S. Naval Ordnance Test Station, China Lake, California 1962.
17. Matsui, H. and Lee, J.H., "On the Measure of the Relative Detonation Hazards of Gaseous Fuel-Oxygen and Air Mixtures," Seventeenth Symposium (International) on Combustion, 1269-1280, 1978.
18. Korovin, L.N., Losev, A., S.G. Ruban and Smekhov, G.D. "Combustion of Natural Gas in a Commercial Detonation Reactor," Fiz. Gor. Vzryva, Vol. 17, No. 3, p. 86, 1981.
19. Smirnov, N.N. and Boichenko, A.P., "Transition from Deflagration to Detonation in Gasoline-Air Mixtures," Fiz. Gor. Vzryva, 22, No. 2, 65-67, 1986.
20. Lobanov, D.P., Fonbershtein, E.G. and Ekomasov, S.P., "Detonation of Gasoline-Air Mixtures in Small Diameter Tubes," Fiz. Gor. Vzryva, 12, No. 3, 446, 1976.

21. Back, L.H., "Application of Blast Wave Theory to Explosive Propulsion," Acta Astronautica, 2, No. 5/6, 391-407, 1975.
22. Varsi, G., Back, L.H. and Kim, K., "Blast Wave in a Nozzle for Propulsion Applications," Acta Astronautica, 3, 141-156, 1976.
23. Kim, K., Varsi, G. and Back, L.H., "Blast Wave Analysis for Detonation Propulsion," AIAA Journal, Vol. 10, October 1977.
24. Back, L.H., Dowler, W.L. and Varsi, G., "Detonation Propulsion Experiments and Theory," AIAA Journal Vol. 21, October 1983.
25. Helman, D., Shreeve, R.P. and Eidelman, S., "Detonation Pulse Engine," AIAA-86-1683, 24th Joint Propulsion Conference, Huntsville, 1986.
26. Eidelman, S., Grossmann, W. and Lottati, I., "Propulsion Applications of the Pulsed Detonation Engine Concept," SAIC Report Number 89/1684, December 31, 1989.
27. Eidelman, S., Grossmann, W. and Lottati, I., "A Review of Propulsion Applications of the Pulsed Detonation Engine Concept," J. Propulsion and Power, Vol. 7, No. 6, November-December 1991 (857-865).
28. Eidelman, S. and Grossmann, W., "Computational Analysis of Pulsed Detonation Engines and Applications," AIAA-90-0460, January 8-11, 1990/Reno, Nevada.
29. Eidelman, S., Grossmann, W. and Lottati, I., "Air-Breathing Pulsed Detonation Engine Concept: A Numerical Study," AIAA-90-2420, July 16-18, 1990/Orlando, Florida.
30. Eidelman, S., Lottati, I. and Grossmann, W., "A Parametric Study of the Air-Breathing Pulsed Detonation Engine," AIAA-92-0392, January 6-9, 1992/Reno, Nevada.
31. Lottati, I., Eidelman, S. and Drobot, A., "A Fast Unstructured Grid Second Order Godunov Solver (FUGGS)," AIAA-90-0649, January 8-11, 1990/Reno, Nevada.
32. Anderson, J. D., Modern Compressible Flow, McGraw-Hill, 2nd Edition, New York, 1982.

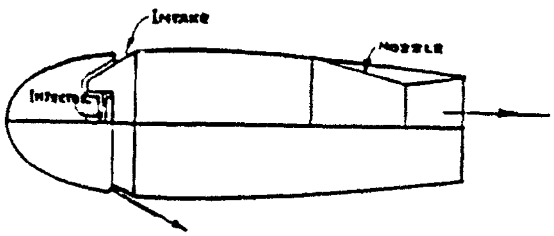


Figure 1. Valveless propulsive duct concept due to Tharjatt.

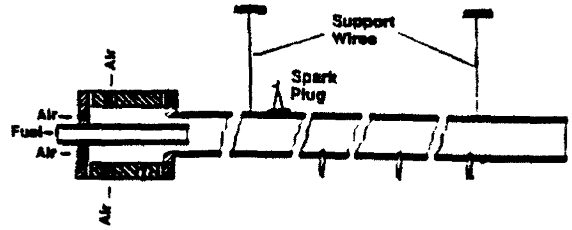


Figure 3. Schematic of the Hoffmann - Nicholls - Krzycki detonation tube experimental apparatus.

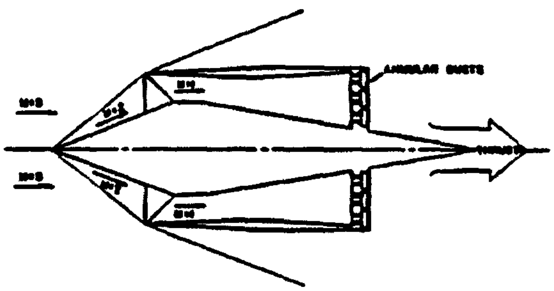


Figure 2. Supersonic, M = 3 conceptualization of the propulsive duct.

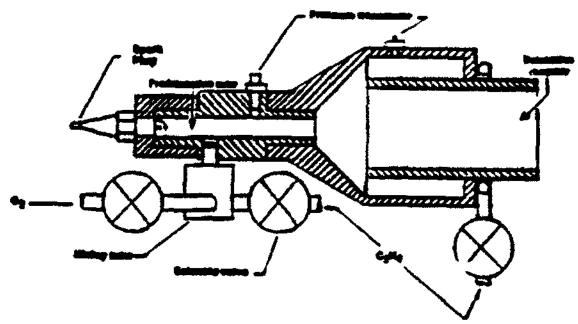


Figure 4. Schematic of the Helman, Shreeve, Eidelman PDE experimental configuration from the NPS studies.

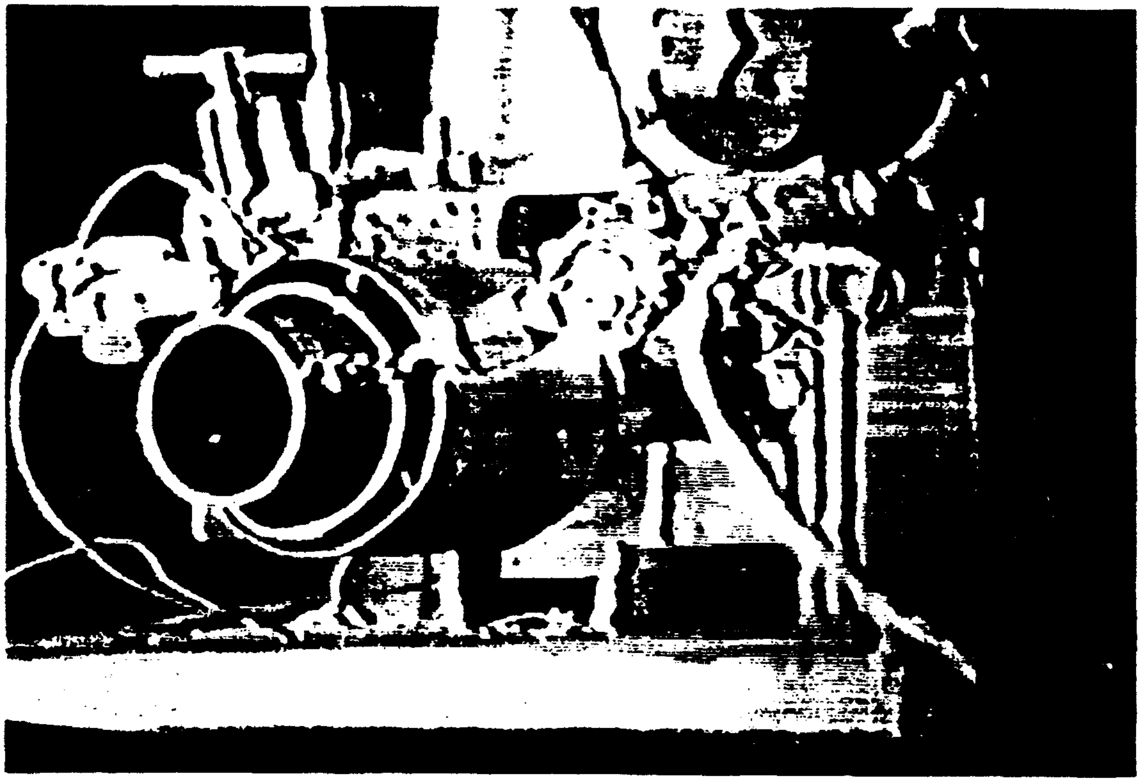


Figure 5. The PDE experimental apparatus used in the NPS studies.

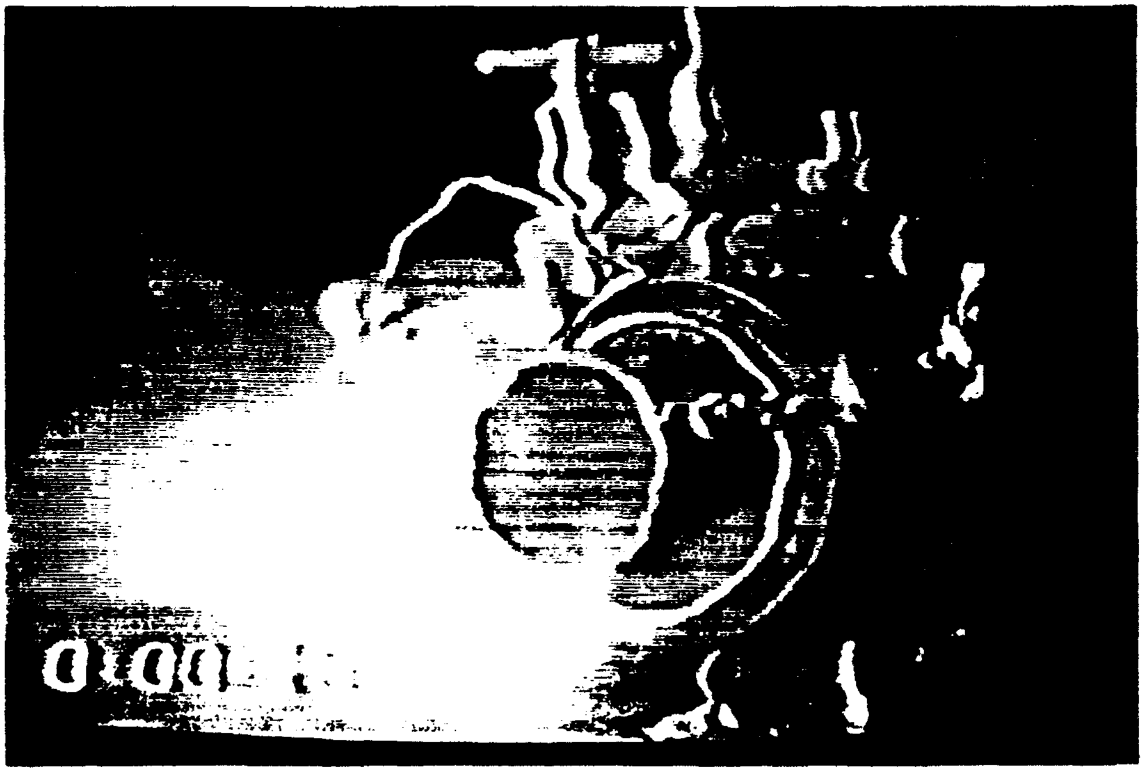


Figure 6. The PDE experiment during repetitive detonation.

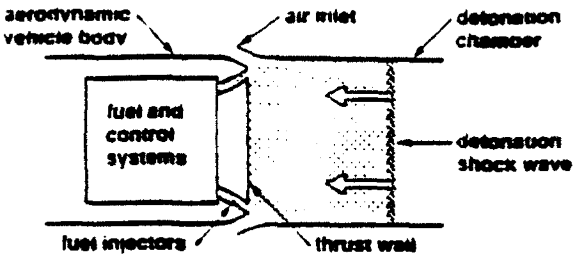


Figure 7. Schematic of the generic PDE.

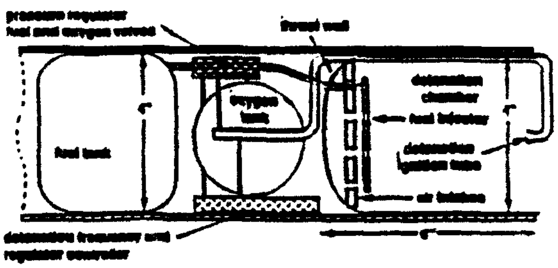


Figure 8. Schematic of PDE/PENAID missile integration.

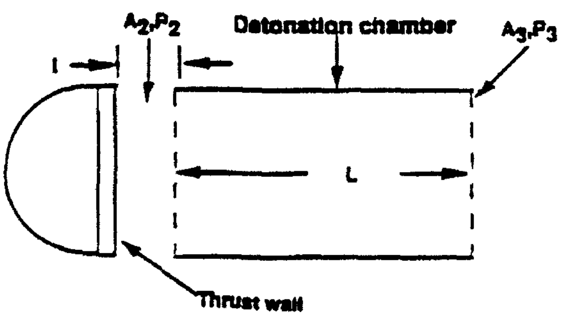


Figure 9. Schematic of PDE describing key sizing variables.

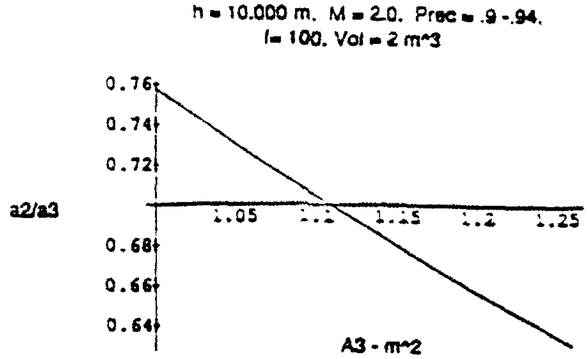


Figure 10. Results for A_2/A_3 as a function of A_3. The results are, for the chosen conditions, independent of pressure recovery.

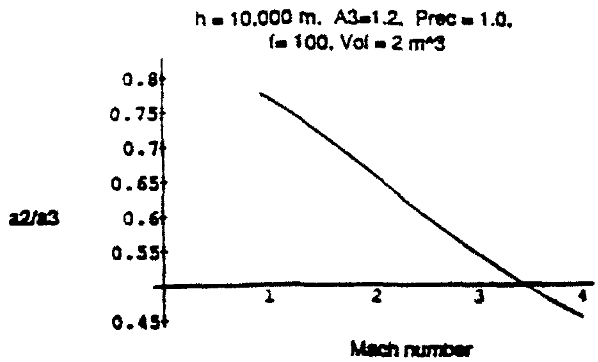


Figure 11. Results for A_2/A_3 as a function of Mach number.

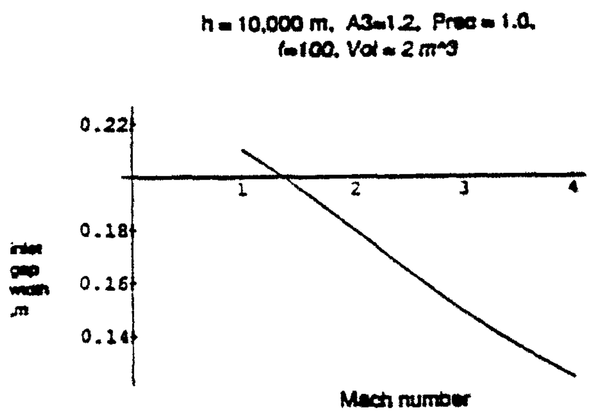


Figure 12. Results for inlet gap width, l, as a function of Mach number.

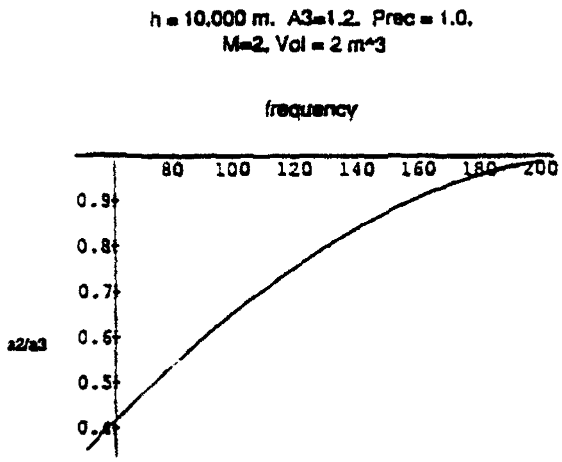


Figure 13. Results for A_2/A_3 as a function of detonation frequency.

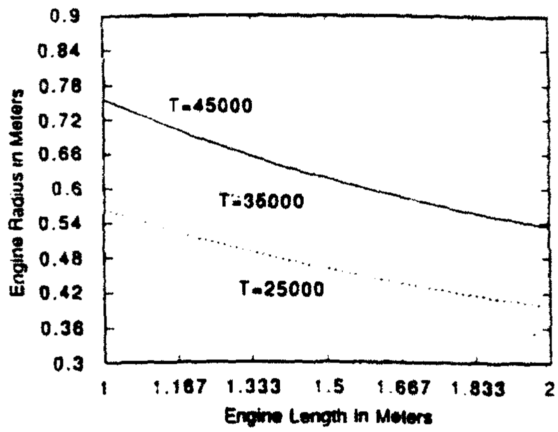


Figure 15. PDE engine radius (cylindrical cross-section) versus engine length.

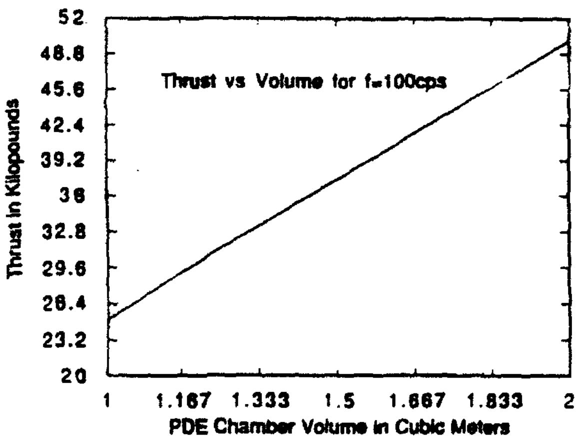


Figure 14. PDE thrust versus detonation chamber volume at a given frequency, f = 100 Hz.

Synthesis of Nanoscale Materials Using Detonation of Solid Explosives

Shmuel Eidelman and Anatoly Altshuler
Science Applications International Corporation
1710 Goodridge Drive
Mail Stop 2-3-1
McLean, Virginia 22102

First International Conference on Nanostructured Materials
Cancun, Mexico
September 21-26, 1992

Synthesis of Nanoscale Materials Using Detonation of Solid Explosives

Shmuel Eidelman and Anatoly Altshuler
Science Applications International Corporation
McLean, Virginia 22102

Abstract

Direct synthesis of nanophase materials in detonations is considered. Article discusses a number of methods that can lead to formation of super saturated states of media that in turn will presipitate as nanoscale particles when the detonation products are quenched in the expansion process. Several examples are given of reactions that will lead to production of nanophase particles of metals, oxides, diamond and other unique materials. It is shown that conditions of nucleation and growth of nanoscale material can be analysed using advanced methods of computer simulation of detonation and blast wave phenomena. A sample of this kind of simulations is given. It is concluded that detonative synthesis of nanophase material can lead to low cost technology that will produce a range of unique materials.

1. Introduction

Recent enhanced interest in nanoscale materials is merited by the discovery of a set of unconventional material properties in the form of particles that are less than 10 nm in size. Anomalous chemical activity, lower critical temperatures of oxidation and sintering, sintering of composite materials with manifold increase in tensile strength, and sintering unique semiconducting and ferromagnetic materials, have all been demonstrated for nanoscale materials. This wide range of applications makes nanosize materials an extremely interesting and important material state that is the subject of intense study by many researchers.

The synthesis of nanoscale materials is accomplished through methods such as ion-sputtering and ion-deposition, laser ablation, evaporation and condensation in a vacuum, solgel, electroprecipitation, and plasma-jet techniques. Each of these techniques has produced 2-10 nm particles of various materials; however, the yield of such processes is extremely low and the cost of materials obtained is very high.

In this article we will consider detonative synthesis, a method of nanosize materials synthesis that offers an alternative to other more costly methods of production. Detonative synthesis is extremely advantageous because it allows very high pressure and temperature conditions to be created using low cost explosive materials and simple processing equipment. The synthesis occurs directly in the plasma created by the detonation wave. Conditions for detonative synthesis can be modified by changing the physical and chemical conditions of the detonation wave and expanding detonative products. For example, for nanoscale diamond powder synthesis, rapid expansion and cooling of the detonation products are required to prevent diamond graphitization. Thus, the explosive charge and atmosphere surrounding it should be designed to create these conditions.

In the following, we will review a range of conditions necessary for nanosize material synthesis that is provided by detonative synthesis methods, and examine their applicability for specific materials.

2. Detonation Waves as Generators of High Energy Density Plasmas

Detonations are reactive wave phenomena in which a reaction is initiated by the shock waves propagating at supersonic speeds through an explosive mixture. This wave consists of a shock wave discontinuity followed by a narrow zone of homogeneous chemical reaction. The shock wave compresses the explosive from its initial state with pressure P_0 and density ρ_0 to the shocked state P_s, ρ_s, with subsequent reaction of explosive in the reaction zone that extends up to the Chapman-Jouguet (CJ) state.

Condensed Explosive Detonations

Table I gives some typical parameters for detonation waves in solid explosives. We can see from this data that temperatures of about 3000°C at pressures of 30 GPa are typical for solid explosive detonations. These parameters create extremely oversaturated conditions for some detonation products. Subsequent ultra-fast quenching can lead to synthesis of nanophase material. Behind the detonation wave reaction zone, temperatures and pressures are high and detonation products will usually contain various active chemical components. It is challenging in this environment to preserve nanosize material from further reaction.

TABLE I

Some Typical Conditions for Detonation of Solid Explosives (1)

	Pressure GPa	Temperature °K	D velocity m/sec
TNT, ρ=1.6 g/cm^3	20.6	2940	6950
RDX, ρ=1.8 g/cm^3	34.7	2590	8750
HMX, ρ=1.9 g/cm^3	39.5	2364	9160
PbN_3, ρ=4.0 g/cm^3	23.1	2660	5000

Multi-Phase Detonations

Multi-phase detonations can cover a range of conditions between gaseous and condensed material detonations. Multi-phase detonable mixtures can be composed of solid or liquid fuel particles dispersed in gaseous oxidizer, solid particles of explosive material dispersed in gas, gaseous explosive mixture mixed with the inert or reactive liquid phase (2), or explosive slurries. All these possible methods of generating detonation waves greatly extend the range of conditions available for material synthesis. It should be noted that there is a difference in the character of condensed explosive detonation and gaseous detonations. With condensed explosives, high rate decomposition reactions usually take place. For gaseous detonations, reactions can be characterized as detonative combustion. Multi-phase detonations can be based on detonative combustion, high rate decomposition, and combinations of these processes.

Nonstandard Regimes for Detonative Reaction

A classical self-sustained detonation wave has a fixed wave structure that moves through the explosive with constant velocity. In a self-sustained detonation, a balance is achieved between the compression work of the shock wave and energy released in the reaction zone. If a self-sustained detonation is possible in a given explosive mixture at given initial conditions, it will propagate with a constant speed.

However, for many important reactive mixtures it is either very difficult or impossible to obtain a self-sustained detonation wave.

Over the last forty years, many nonstandard detonation regimes have been discovered that significantly reduce the restrictive limitations of the classical self-sustained detonation. The following is an incomplete list of the detonation regimes that significantly deviate from the classical self-sustained detonation wave:

a. Transient detonation (forms when a deflagration wave undergoes transition to detonation);
b. Overdriven detonation (compression work of the leading shock is partially sustained by an external source of energy);
c. Spinning detonation (formed by small number of detonative combustion fronts that propagate through the mixture by spinning);
d. Multi-layer detonation (propagates in layers of explosives where the detonation wave in one layer can lead to lateral initiation of an overdriven detonation wave in the adjacent layer);
e. SWACER (Shock Wave Amplification by Coherent Energy Release) detonation;
f. Light supported detonation (detonation front is supported by a laser beam heating the area behind the shock front).

All these possible regimes for initiating and sustaining detonation waves allow substantial flexibility in adapting a detonative process for the purpose of material synthesis.

3. Detonative Synthesis Chemistry for Nanophase Materials

The elementary composition of known explosives is quite limited. The most common class, CHNO explosives, produces only one condensed phase under normal thermodynamic conditions – ultra fine carbon (1):

$$C_3H_6N_6O_6(RDX) \underline{\rightarrow} 3H_2O + 1.49CO_2 + 0.022 \cdot CO + 3N_2 + 1.49C_{(s)}$$
$$C_7H_5N_3O_6(TNT) \underline{\rightarrow} 2.5H_2O + 1.66CO_2 + 0.188 \cdot CO + 0.001 \cdot NH_3 + 1.5N_2 + 5.15C_{(s)}$$
$$C_4H_8N_8O_8(HMX) \underline{\rightarrow} 4 \cdot H_2O + 2 \cdot CO_2 + 0.008 \cdot CO + 4N_2 + 2 \cdot C_{(s)}$$

These reactions have the following yield limits for solid phase carbon: 9% for RDX or HMX, and 29% for TNT.

More "exotic" BCHNO explosives can decompose, which produces solid BN or B_2O_3. For example, the powerful explosive $B_{10}H_{100}C_{5.75}N_{15}O_{30}$, decomposes with the 26% yield of BN by weight, while less hydrogenized $B_{10}H_{18}C_{5.75}N_{15}O_{30}$ produces primarily B_2O_3 with 34% yield.

From the point of view of chemical productivity, the most promising class of explosives is presented by acetylides and azides. For example, explosive decomposition of silver acetylide ($Ag_2C_2 \underline{\rightarrow} \% Ag + 2C + 87 kcal/mol$) generates a 90% silver yield. A more powerful explosive decomposition of $Ag_2C_2 \cdot AgNO_3 \underline{\rightarrow} 3Ag(vapor) + CO_2 + CO + 0.5N_2 + 185 kcal/mol$, gives 80% silver yield but much finer dispersity is expected. The decomposition of silver acetylides is interesting to compare with a silver azide explosion, $2Ag(N_3) \underline{\rightarrow} 2Ag(v) + 3N_2$, with a respective yield of silver on

the order of 72% by weight. Over two dozen metals form explosive azides, while explosive acetylides are less common. Among the most interesting azides for nanosize powder production are explosive azides of cobalt, gold, strontium, and platinum. The main challenge in producing nanophase metals by explosive decomposition of azides or acetylides will be to assure rapid quenching of nanoscale phase components of the explosive products.

Loaded Explosive Synthesis

Explosive compositions are unknown for some chemical elements, as in the case of aluminum. The most obvious solution is to mix the explosive carrier with the powder or liquid form of the desired chemical. There is already a substantial history of adding aluminum powder to explosives in order to increase their performance. It has been established that at a grain size of several microns, aluminum does not have time to sublimate in the detonation wave reaction zone; thus, it will not affect the reaction rates. On the other hand, detonation energies and temperatures are high enough to evaporate a substantial amount of additive. In order to overcome the diffusion barrier, we are considering mixing a melted explosive carrier with a liquid aluminum compound like $AlBr_3$, which has a melting point of 97°C and comparatively low evaporation energy and temperature. Aluminum azide is also a possibility.

The same approach can be implemented in the loaded explosive synthesis of the nanoscale Hf. In this case, we can use detonation mixture of $Hf(BH_4)_4$ and an explosive carrier. Similarly, Ir can be produced using an IrF_6 load; Pu using a PuF_6 load; Re using a ReF_6 load; U using an UF_6 load; W using a WCl_6 load; V using a VF_5 load; Ti by means of a $TiCl_4$ load, etc. The reduction of metals in all these cases is taking place both physically, as the result of shock-temperature dissociation of molecules, and chemically, by ionized hydrogen and, in some cases, lithium vapors.

For carbon synthesis, loading the explosives cited above with benzol (C_6H_6), 1-hexadecen ($C_{16}H_{32}$), hexacozan ($C_{26}H_{54}$), dibenzyl ($C_{14}H_{14}$) etc., can greatly increase the yield of carbon without substantially diminishing the energetic characteristics of detonation. For example, a mixture of benzol with HMX on mol to mol basis will decompose in the detonative reaction as follows:

$$C_6H_6 + C_4H_8N_8O_8 \underline{\rightarrow} 7H_2O + 0.5CO_2 + 4N_2 + 9.5C_{(S)}$$

This reaction yields 30% by weight of solid carbon that has a potential to be preserved in nanoscale form.

All these examples illustrate that the loading of explosives for nanophase material synthesis expands the range of opportunities beyond the synthesis that results from the detonative decomposition of explosives.

Phase Composition of Synthesis Products

The crystalline structure of nanoscale powders obtained from detonation generally corresponds to high-pressure modifications of the solids. This is the result of high temperature and high pressure conditions in the detonation wave reaction zone and subsequent ultra-fast quenching and cooling of detonation products. In the case of carbon, diamond is formed. The phase diagram for carbon shown below easily illustrates this point. Area marked with number 1 on the phase diagram reflects parameters typical for detonation of HMX, while the area marked with 2 corresponds to detonation of TNT. It is quite obvious from Figure 1 that the detonation of TNT cannot produce diamond, while the detonation of HMX brings all condensed carbon into diamond form.

The same situation occurs with the synthesis of BN, when explosive decomposition of boron azide $B(N_3)_3$ produces hexagonal modification of BN, while powerful BCHNO explosives can produce BN with cubic sfalerite structure. Other compounds that can be obtained include interesting compositions such as ZrO_2, HfC, and WC, sometimes in their metastable modifications. Much more diverse are crystalline modifications of nanoscale metals. In cases like Gadolinium (Gd) and Samarium (Sm), five different structure modifications could be obtained as a result of different experimental conditions.

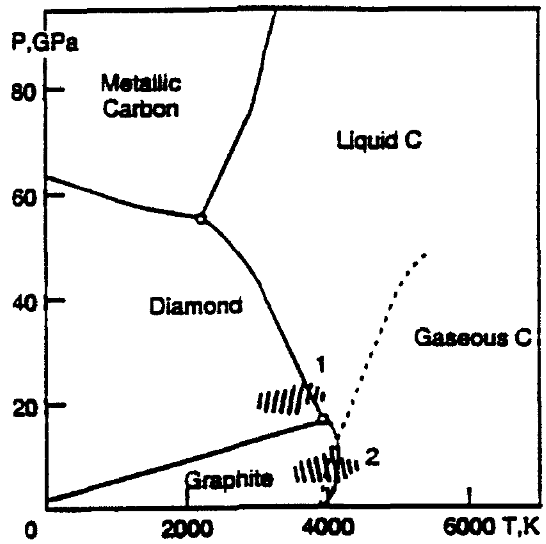


Figure 1. Carbon phase diagram schematics.

Nucleation and Growth of Nanophase Material Behind the Detonation Waves

We have discussed above the detonation wave structure in solids. We made important assumptions in our previous analysis regarding chemical equilibrium and physical stationarity of the processes on detonation front. The characteristic time of typical explosive decomposition reactions under detonation conditions in solid explosives lies in the range of $10^{-11} \div 10^{-12}$ sec. As we noted above, the characteristic time length of the reaction zone for detonations in solids is $10^{-7} \div 10^{-8}$ sec. This difference in time scales allows us to consider reactions behind the detonation front as equilibrium decompositions. Phenomenologic criteria of nucleation stationarity according to V. Shreidman (2) can be presented as follows:

$$\nu \leq \left(\frac{W}{T}\right)^{-1} \rho \sigma^2 / \eta^3 \qquad (1)$$

where ν - characteristic frequency of external forces; W - activation energy of nucleation; T - temperature in energetic units; ρ and η - density and viscosity of gases; σ - surface tension coefficient for nuclei.

Following are Fol'mer theory (3) we present activation energy through thermodynamic parameters:

$$W = \frac{16\pi}{3} \frac{\sigma^3 V^2}{(T \ln P / P_e)^2}. \qquad (2)$$

Here, P - partial pressure in gaseous precipitous phase; P_e - equilibrium pressure of saturation for condensate at given T; V - atomic volume in condensed phase.

For the conditions typical for diamond condensation in the process of detonative synthesis, the barrier of nucleation at 100 kbar pressure and 3000°k temperature behind the detonation front, is W = $13 \cdot 10^{-12}$ erg. Criteria (1) in this case gives: $\nu \leq 10^{13} \div 10^{14} Hz$. Considering the time span of the detonation wave reaction zone ($10^{-7} \div 10^{-8}$ sec), we can assume stationarity of diamond nucleation. Calculations made for metals and some inorganic compounds lead to the same conclusion.

In accordance with the stationary approximation, the nucleation rate can be presented as follows (3):

$$I = \frac{2\alpha P^2 V \sigma^{1/2}}{(2\pi m T)^{1/2} T^{3/2}} \exp\left(-\frac{W}{T}\right) \qquad (3)$$

α - condensation coefficient, m - atomic mass of condensate. For our reference case of diamond nucleation, calculation using equation (3) gives: $I \approx 10^{21} \frac{nuclei}{\sec \cdot cm^3}$

The diameter of critical nuclei can be estimated from activation barrier: $D = \sqrt{\frac{3W}{\pi\sigma}}$. For diamond it gives D ~ 5Å.

4. Solid Explosive Charge Detonation in a Confined Volume

Experimentally developed conditions for diamond powder synthesis rely on the multi-layered detonation of several explosives and inert material. This system undergoes a complex detonation under conditions that are overdriven for the explosive producing diamond powder, and are standard for the driver detonation with some complex multi-dimensional expansion into the surrounding media. The details of the detonation process in this system have never been studied computationally, but experimental methods indicate that very specific conditions are required. It is known that the end result of this process is extremely sensitive to conditions of the multi-layered detonation. Currently, it is not clear what variables control particle sizes, or the maximum amount of free carbon released during the detonative combustion process that can be synthesized into diamond. Experimental work in this field is sketchy; numerical analysis of this complex process will enable us to understand the sensitivity to the basic parameter variations controlling diamond synthesis. Below are the results of numerical simulation of detonation and detonation products expansion for a composite TNT/RDX charge detonated in a 1 M^3 chamber. This simulation will give the conditions of the detonative products at various stages of expansion that determines the environment prevalent in the detonative synthesis.

In Figure 2 schematics of the blast sphere cross section are shown with the solid explosive charge located at the sphere's center. The inner volume of the sphere is 1 M^3. Solid explosive is a composite charge formed from a TNT main charge with the layer of RDX around it. Detonation of a high energy RDX layer leads to the formation of an overdriven detonation wave in the main charge. Because the problem is symmetric, it is sufficient to simulate one quarter of the sphere volume to describe the full range of blast interaction that will occur for this condition. To increase the simulation's accuracy, we have divided the numerical modeling in the near field and global blast simulations. For the near field, a square grid with DR = DX = 1 mm was used to describe a region 10 cm x 10 cm containing the solid explosive charge. The simulation results from the near field region are mapped on the larger computational domain, which includes the inner wall of the blast sphere. For higher resolution and computational efficiency, we

have used structured/unstructured grids to describe the sphere's inner volume. The mathematical formulation and numerical method for the solution used in the near field are described in detail in Reference 2. These computational techniques are implemented in the MPHASE code. The model and numerical methods used for simulations in the computational domain shown in Figure 2 are described in Reference 5. These computational techniques are implemented in AUGUST code. Both MPHASE and AUGUST have been validated for the range of detonation and strong shock wave reflection and diffraction problems.(6)

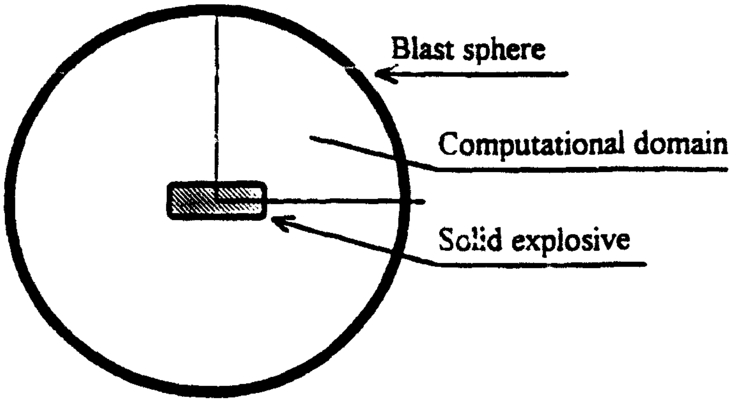


Figure 2. Schematics of the blast sphere cross section with the solid explosive charge. The computational domain is covering the upper right quadrant.

In Figure 3, simulation results for the near field region are shown as pressure density and temperature contour plots for an instant of time when the detonation wave is at 2 mm distance from the right edge of the charge. $t = 0$ is the time of detonation wave initiation in a solid explosive. Pressure and temperature contour plots are shown using a linear scale. In Figure 3 we observe propagation of the complex detonation front through the composite charge and the initial stages of detonation product expansion. The outer layer of the RDX leads to the formation of an overdriven detonation wave in the TNT charge that has shorter reaction zone, higher wave speed, higher temperatures, and higher pressures as compared with a homogeneous TNT charge detonation. The maximum temperature is reached in the air strata located in the immediate vicinity of the charge. This temperature maximum is created by a strong shock wave produced by expanding detonation products in air. The following conditions are reached at the detonation wave front in the TNT charge: $P = 62.6$ GPa; $T = 6000°C$; $\rho = 2900$ kg/m^3. It should be noted that because of high resolution of the numerical scheme we are simulating the Von Neumann spike of the detonation wave front, where the pressure is considerably higher than at the Chapman-Jouguet point.

When the shock wave reaches the edges of the computational domain for the near field simulation, the simulation results are mapped to the grid of the global domain shown in Figure 2 and are continued on larger grid. In Figure 4 pressure and temperature contour plots are shown for three consecutive instances of time for the global domain simulation. In Figure 4a results are shown at $t = 0.05$ μsec, shortly before the detonation products reached the walls of the sphere. Here we can observe significantly lower pressures as compared with Figure 3 values due to strong expansion; however, the propagating shock is leading to considerable heating of the surrounding air. In Figure 4b pressure and temperature contour plots are shown at some stage of the wave front reflection from the inner wall of the blast sphere. The average pressures and temperatures are significantly lower; however, several focus points are created during the reflection that have significantly higher pressure and temperature values. In Figure 4c, the shock wave complex is converging towards the blast sphere center, with significant amplification of the shock strength and temperature at the front. It is obvious that this system of shock waves will undergo a number of reflections. focusing, and expansions until quiescent conditions are reached in the blast sphere.

The simulations illustrated above will provide the global conditions in the blast chamber as a function of time. This information can be used for the nucleation simulations of the material behind the shock front, and estimates of possible phase transformation or reaction of the newly formed material. As a result of this multi-step approach, we can consider all the stages of the detonative synthesis process that are important for nanoscale material formation. This approach will allow us to minimize the number of experiments, understand the physics of detonative synthesis, and control the quality and yield of nanoscale materials produced experimentally.

5. Conclusions

Detonative synthesis of nanoscale material is a new technology and the nature of this process is widely unexplored. More studies should be done in addressing chemical and phase transformations under extreme and fast changing conditions in waves of detonation, shock and rarefraction. An unlimited array of elements and compounds, as well as their structural modifications (some highly metastable), is attainable through such processing.

Detonation synthesis combines the best features of traditional nanophase material technology -- the most effective generation of hot plasma and vapors, and fast quenching of a condensing product. The most unique feature of the process is the extreme density of the generated plasmas, which makes them highly supersaturated in regard to pressure and temperature.

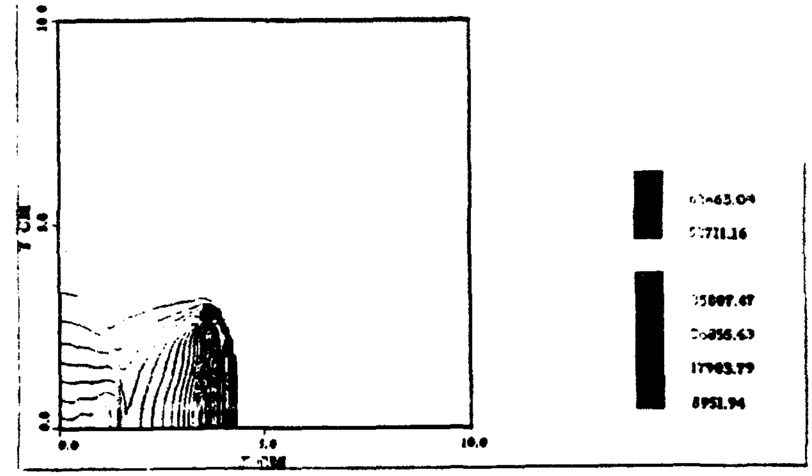
Detonative technology has promising industrial prospects, due to very low production cost and the unique materials it yields. As a reference we can use ultra-fine carbon. In this case common nanotechnology produces carbon black; detonative synthesis diamond. These factors are completely changing the traditional view of nanomaterials applications. (7)

6. Acknowledgment

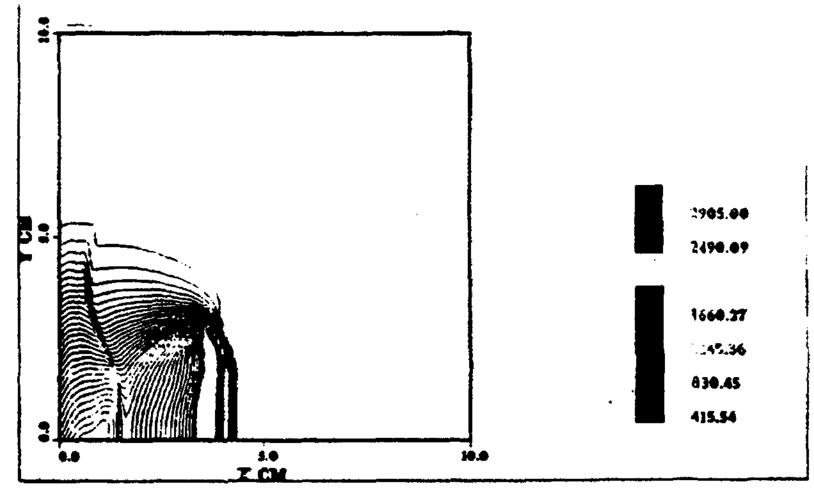
The work reported here was partially supported by ONR under Contract #N00014-92-C-0168. The authors would like to thank Dr. L.T. Kabacoff for his interest in this project.

7. References

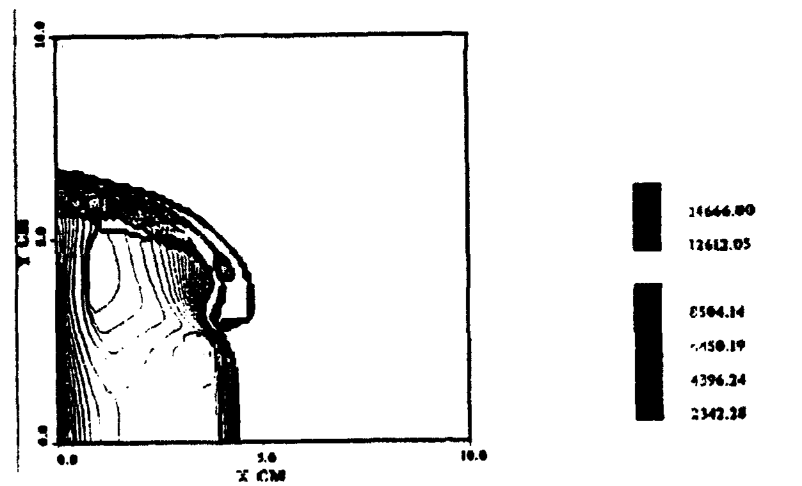
1. C.L. Mader, "Numerical Modeling of Detonations," University of California Press Ltd., London, England. (1979).
2. V. Shreidman, Zhurn Eksperim, Teor. Fiz., v 91, 8, p 520, (1986).
3. D. Fedoseev, Uspekhi Khimii, V 53, p 7 53, (1983).
4. S. Eidelman and X. Yang, "Detonation Wave Propagation in Variable Density Multi-Phase Layers," AIAA 92-0346, 30th Aerospace Sciences Meeting, Reno, NV, Jan. 6-10, 1992.
5. I. Lotatti, S. Eidelman, A. Drobot, "A Fast Unstructured Grid Second Order Godunov Solver (FUGGS)," AIAA 90-0699, 28th Aerospace Sciences Meeting, Reno, NV, Jan. 8-11, 1990a.
6. I. Lottati and S. Eidelman, "Second Order Godunov Solver on Adaptive Unstructured Grids." Proceedings of the 4th International Symposium on Computational Fluid Dynamics, Davis, CA, September 1991.
7. A. Altshuler and J.L. Sprague, "The Synthesis, Properties, and Applications of Diamond Ceramic Materials." 41st Electronic Components Technology Conference, Atlanta, Georgia, May 1991.



a. Pressure



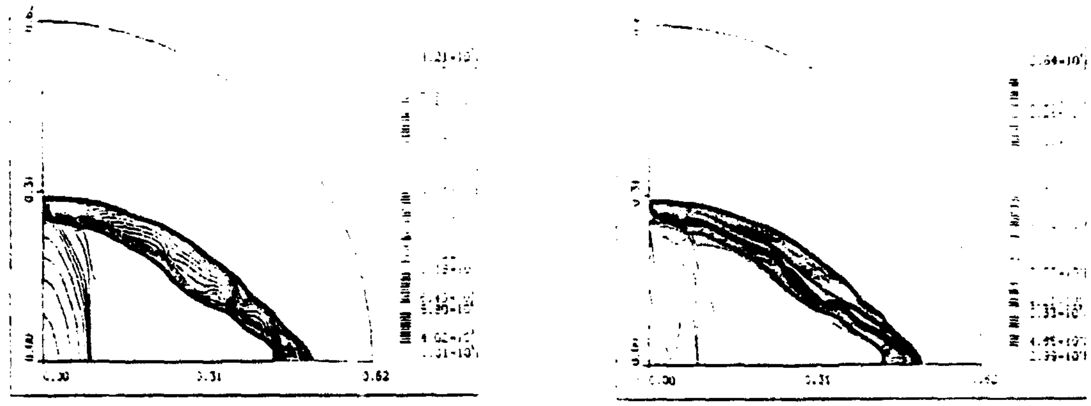
b. Density



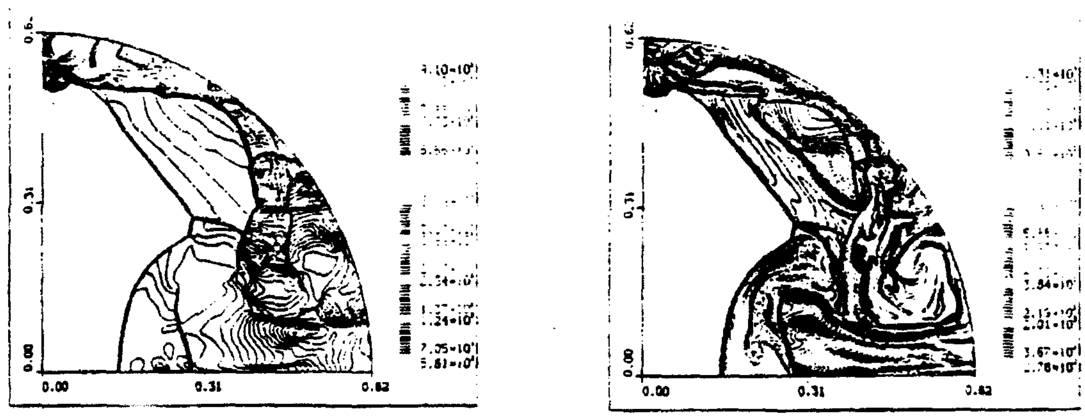
c. Temperature.

Figure 3. Pressuure, density and temperature contour plots for a composite charge detonation t = 0.01 msec.

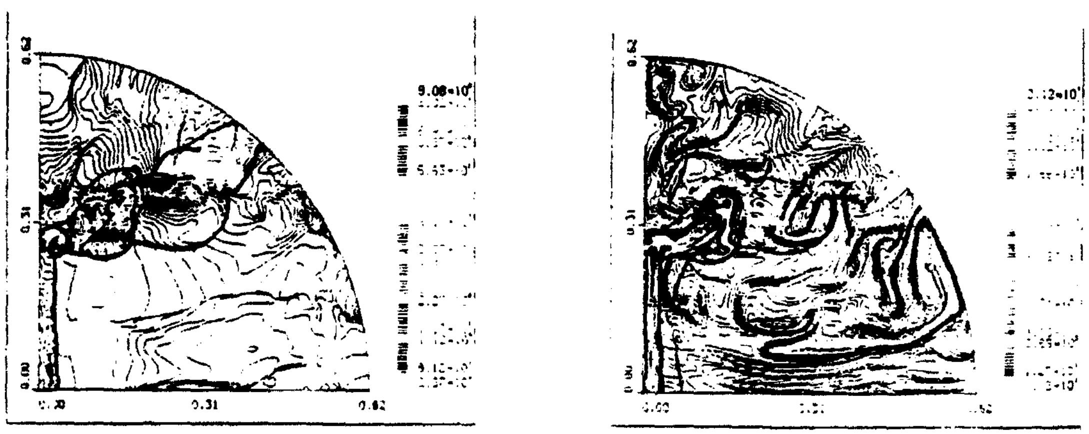
Pressure | Temperature



a. t = 0.05 msec



b. t = 0.2 msec



c. t = 0.3 msec

Figure 4. Simulation of blast wave reflection from the inner wave of a blast sphere.

Detonation Wave Propagation in Combustible Mixtures with Variable Particle Density Distributions

Shmuel Eidelman* and Xiaolong Yang*
Science Applications International Corporation. McLean. Virginia 22102

Abstract

A mathematical model is presented describing a physical system of detonation waves propagating in a solid particle/air mixture with a wide range of solid-phase concentrations. The mathematical model was solved numerically using the Second Order Godunov method, and numerical solutions were validated for detonation waves propagating in mixtures with concentrations of solid phase from 0.75 kg/m^3 to 1000 kg/m^3. Numerical solution was obtained for detonation waves propagating in a system consisting of clouds with a small concentration of particles and a ground layer in which solid particle densities are three orders of magnitude larger than in the cloud. Three different particle concentration distributions in the ground layer were simulated and compared in terms of detonation wave structure and parameters.

Introduction

When combustible particles are intentionally or unintentionally dispersed into the air, the resulting mixture can be detonable. Formation of this potentially explosive dust environment and the properties of its detonation are of significant practical interest in view of its destructive or creative effects.

*Research Scientist, Applied Physics Operation.

The experimental and theoretical study of these phenomena until now has addressed only homogenous particle/oxidizer mixtures. However, intentional or accidental processes of the explosive dust dispersion will always lead to inhomogeneous particle density distribution. Some industrial methods of explosive-forming rely on detonation of explosive powder. This powder can be deposited as a thin layer over the surface area of the forming metal, with some remaining concentration in the vicinity of the layer. The phenomenology of detonation wave initiation and propagation in this environment is the main subject of this paper.

When the detonation wave is generated in a homogeneous mixture by a "direct initiation," it starts with a strong blast wave from the initiating charge. As the blast wave decays, combustion of the reactive mixture behind its shock front starts to have a larger role in support of the shock wave motion. When the initial explosion energy exceeds some critical value, transition to steady-state detonation occurs.[1-4] In explosive dust mixtures with a nonuniform distribution of particle density, the initiation dynamics are significantly more complicated. The critical initiation energy sufficient for one of the explosive particle density strata regions is not necessarily adequate for other regions. Also, when there is a significant variation in density between the different layers (regions) of the mixture, steady detonation in one layer can result in an overdriven detonation in an adjacent layer. Our paper demonstrates that the phenomenology of these interactions is distinctly different from the classical studies of multilayer detonations in gases. This is primarily because the energy content of adjacent layers in a typical multigas layer experiment[5] varies by a factor of two or four, whereas the energy content in explosive dust/air mixtures can vary by several orders of magnitude.

In this paper we use detailed numerical simulation to study the initiation dynamics and propagation phenomenology for a general case of explosive dust dispersion. We will consider particle density variation from 1000 kg/m^3 in the ground layer to 0.5 kg/m^3 or 0 for the upper edges of the cloud. The effects of variation of the cloud density on detonation wave parameters

will be examined for different cases of cloud particle density distribution. When possible, the results of computer simulations are validated in comparison with experimental and theoretical studies.

The outline of this paper is as follows. Section 2 gives a description of a mathematical model that includes governing conservation equations for two phases and the constitutive laws. We describe the model for a particle-gas interaction, combustion, and equation-of-state for gas phase. The numerical integration technique for solving the mathematical model will also be outlined. In Section 3, we present our numerical simulation results. We first validate our model by comparing one-dimensional detonation wave simulation with available experimental results. We then give the two-dimensional simulation for detonation wave propagation in combustible particles/air mixtures with variable particles density distribution. Concluding remarks are given in Section 4.

Mathematical Model and the Numerical Solution

The mathematical model consists of conservation governing equations and constitutive laws that provide closure relations for the model. The basic formulation adopted here follows the two-phase fluid dynamics model presented in the text by Kuo.[6] The approach assumes that there are two distinct continua, one for gas and one for solid particles, each moving at its own velocity through its own control volume. The sum of these two volumes represents an average mixture volume. With these assumptions, distinct equations for continuity, momentum, and energy are written for each phase. The interaction effects between the two phases are accounted for by the source terms on the right-hand side of the governing equation. The following is a short description of the two-phase flow model used in our study, with conservation equations written in Eulerian form for two-dimensional flow in Cartesian coordinates:

Continuity of gaseous phase

$$\frac{\partial \rho_1}{\partial t} + \frac{\partial(\rho_1 u_g)}{\partial x} + \frac{\partial(\rho_1 v_g)}{\partial y} = \Gamma \qquad (1)$$

Continuity of solid-particle phase

$$\frac{\partial \rho_2}{\partial t}+\frac{\partial(\rho_2 u_p)}{\partial x}+\frac{\partial(\rho_2 v_p)}{\partial y}=-\Gamma \tag{2}$$

Conservation of momentum of gaseous phase in x direction

$$\frac{\partial(\rho_1 u_g)}{\partial t}+\frac{\partial(\rho_1 u_g^2+\phi p_g)}{\partial x}+\frac{\partial(\rho_1 u_g v_g)}{\partial y}=-F_x+\Gamma u_p \tag{3}$$

Conservation of momentum of gaseous phase in y direction

$$\frac{\partial(\rho_1 v_g)}{\partial t}+\frac{\partial(\rho_1 u_g v_g)}{\partial x}+\frac{\partial(\rho_1 v_g^2+\phi p_g)}{\partial y}=-F_y+\Gamma v_p \tag{4}$$

Conservation of momentum of solid-particle phase in x direction

$$\frac{\partial(\rho_2 u_p)}{\partial t}+\frac{\partial(\rho_2 u_p^2)}{\partial x}+\frac{\partial(\rho_2 v_p u_p)}{\partial y}=F_x-\Gamma u_p \tag{5}$$

Conservation of momentum of solid-particle phase in y direction

$$\frac{\partial(\rho_2 v_p)}{\partial t}+\frac{\partial(\rho_2 u_p v_p)}{\partial x}+\frac{\partial(\rho_2 v_p^2)}{\partial y}=F_y-\Gamma v_p \tag{6}$$

Conservation of energy of gas phase

$$\frac{\partial(\rho_1 E_{gT})}{\partial t}+\frac{\partial(\rho_1 u_g E_{gT}+u_g\phi p_g)}{\partial x}+\frac{\partial(\rho_1 v_g E_{gT}+v_g\phi p_g)}{\partial y}=$$

$$\Gamma\Big(\frac{u_p^2+v_p^2}{2}+Echem+C_sT_p\Big)-\Big(F_x u_p+F_y v_p\Big)-\dot{Q} \tag{7}$$

Conservation of energy of solid-particle phase

$$\frac{\partial(\rho_2 E_{pT})}{\partial t} + \frac{\partial(\rho_2 E_{pT} u_p)}{\partial x} + \frac{\partial}{\partial y}(\rho_2 E_p v_p) = \dot{Q} + (F_x v_p + F_y v_p)$$

$$-\Gamma\Big(\frac{u_p^2 + v_p^2}{2} + Echem + C_s T_p\Big) \qquad (8)$$

Conservation of number density of solid-particle

$$\frac{\partial N_p}{\partial t} + \frac{\partial(N_p u_p)}{\partial x} + \frac{\partial(N_p v_p)}{\partial y} = 0 \qquad (9)$$

In the above equations, we have the following definitions and constitutive laws:

Phase densities

$$\rho_1 = \phi\rho_g, \qquad \rho_2 = (1-\phi)\rho_p \qquad (10a)$$

and fractional porosity

$$\phi = 1 - \frac{N_p M_p}{\rho_p} = \frac{\text{Volume of void}}{\text{total volume}} \qquad (10b)$$

where M_p is the mass of each particle and ρ_p is the solid-particle density.

Total internal energy of gaseous phase

$$E_{gT} = E_g + \frac{1}{2}(u_g^2 + v_g^2) \quad \text{and} \quad E_g = E_g(p_g, \rho_g) \qquad (11)$$

where $E_g(p_g, \rho_g)$ is the equation-of-state for gas phase, which will be discussed later.

Total internal energy of solid-particle phase

$$E_{pT} = E_p + \frac{1}{2}(v_p^2 + v_p^2) \quad \text{and} \quad E_p = \text{Echem} + C_s T_p \qquad (12)$$

In order to close the above system of conservation equations, it is necessary to define certain criteria and interaction laws between the two phases, which include mass generation rate, Γ, drag force between particles and gas, F_x, F_y, and the interphase heat transfer rate $\dot{Q}$. The model for particle and gas interaction and particle combustion that results in the constitutive relation for the conservation equations is explained in detail in the next subsection.

Model for a Particle Gas Interaction and Combustion

Presently, the physics of the energy release mechanisms in solid-particles/air mixtures is not clearly understood. This can be attributed to the obvious difficulties of making a direct nonobtrusive measurement in the optically thick environment typical for this system. In the experimental and theoretical work done for the grain dust detonation conditions,[7] it was demonstrated that the volatile components released by the particle heated behind the shock front play a major role in determining the detonability limits of the mixture. Eidelman and Burcat[8] successfully applied a combination of fast evaporation and aerodynamic shattering mechanisms to simulate a two-phase detonation process.

The chemical processes of a single particle combustion, which mainly occur in the gaseous phase, are significantly faster than the physical processes of particle gasification or disintegration. Thus, in the multiphase mixtures, the rate of energy release will be mostly determined by physics of particle disintegration. It is very difficult to describe the details of particle disintegration in the complex environment prevalent behind the shock or detonation wave. For example, Reinecke and Waldman[9] defined five different disintegration regimes for a relatively simple environment of water droplets passing through a weak shock. Fortunately, in most cases of multiphase detonation, only the main features of the particle disintegration dynamics need to be captured to describe the phenomena. For example, Eidelman and Burcat[10] used simple models for particle evaporation and shattering to obtain simulation results that compared very favorably with experimental data. Because of

our inability to resolve the particle disintegration problem in all its complexity, the validation of the model against known experimental data is essential.

In this paper, we consider solid particles consisting of explosive material. Explosive material contains fuel and oxidizer in a passive state at low temperature; however, when the temperature rises the fuel and oxidizer react, leading to detonation or combustion. The intiation of reaction for explosives occurs at relatively low temperature. For example, TNT will detonate when heated to the temperature[11] of 570°C. Only particles larger than a critical detonation size can detonate directly when initiated by a shock wave. Here, consider particles smaller than 4 mm in diameter that will not detonate when heated, but will burn when the temperature on the particle surface reaches a critical value. Since the heat conduction inside the explosive material is relatively slow, the process of particle heating needs to be resolved in detail. Our simulations numerically solve the temperature field in the particles at every step of numerical integration of the global conservation equations. The explosive particle combustion model examined in this paper assumes that the fraction of the particle that reaches the critical temperature will burn instantaneously.

Energy transfer by convection and conduction is simulated by solving the unsteady heat conduction equation in each computational cell at each time step. Assuming a particle's temperature to be a function of time and radial position only, the unsteady heat conduction equation may be transformed to:

$$\frac{d^2w}{dr^2} = \frac{1}{\alpha}\frac{dw}{dt} \tag{13}$$

subject to the boundary conditions:

$$w = 0 \quad \text{at} \quad r = 0, \quad t > 0$$

$$k\frac{dw}{dr} + (h - \frac{1}{R})\, w = hRT_g \quad \text{at} \quad r = R, t > 0 \tag{14}$$

where

$w(r,t) = rT(r,t)$
r = radial position
$T(r,t)$ = temperature
R = particle radius
T_g = temperature of surrounding gas
k = thermal conductivity of particle
h = convective heat transfer coefficient

The Nusselt number, used to find h, is given by an empirical relation given by Drake.[12] The gas viscosity is derived from Sutherland's Law. The gas thermal conductivity is calculated by assuming a constant Prandtl number. Finally, the boiling temperature at a given pressure is derived from the Clapeyron-Clausius equation under the following assumptions: 1) phrasing-constant latent enthalpy of phase-change; 2) the vapor obeys the ideal equation-of-state; and 3) the specific volume of the solid/liquid is negligible compared to that of the vapor. A critical temperature is also employed to serve as an upper limit to the boiling point, regardless of pressure.

Equation 13 with boundary condition 14 can be numerically integrated using either implicit or explicit schemes.

Since the particle radius R becomes very small due to evaporation, the implicit Crank-Nicolson algorithm is used because of its stability properties and its second order temporal and spatial accuracy. Using the Crank-Nicolson scheme to predict the particle temperature profiles at times t_1 and t_2 permits easy calculation of the total energy exchange $\dot{Q}$ between t_1 and t_2, due to convection and conduction.

Knowledge of the particle temperature profile also allows the precise determination of the quantity of the mass to transfer from the particle to the gas Γ. Once any point at a radial location $0 \leq r \leq$ R has a temperature exceeding the boiling temperature, the entire mass between r and R is transferred to the gas phase in one time step. In so doing, an energy equal to the product of the mass lost and the particle intrinsic energy is transferred by the particle to the gas.

The interphase drag force Fx, Fy is determined from the experimental drag for a sphere, as presented by Schlichting.[13]

$$F_x = \left(\frac{\pi}{8}\right) N_p \rho_g C_D |\mathbf{V}_g - \mathbf{V}_p| (u_g - u_p) R^2 \tag{15}$$

where

$$C_D = \begin{cases} \frac{24}{Re}\left(1 + \frac{Re^{2/3}}{6}\right) & \text{for Re} < 1000; \\ 0.44 & \text{for Re} > 1000 \end{cases} \tag{16}$$

and $Re = \frac{2R|V - V_p|}{\mu_g}$, R is radius of particle, and μ_g is gas viscosity at temperature of $T_{film} = \frac{1}{2}(T_g + T_p)$.
Similarly, the formulae for Fy is

$$Fy = \frac{\pi}{8} N_p \rho_g C_D |\mathbf{V}_g - \mathbf{V}_p| (v_g - v_p) R^2 \tag{17}$$

Equation-of-State for Detonation Products

To close the system of governing equations, one needs a constitutive relation between pressure, temperature, and energy for gas phase, which is an equation-of-state. This study uses the Becker-Kistiakowsky-Wilson (BKW) equation-of-state,[14,15] that is,

$$p_g V_g / \bar{R} T_g = 1 + x e^{bx} \tag{18}$$

where

V_g = volume of gas phase
p_g = pressure of gas phase
T_g = temperature of gas phase
$\bar{R}$ = universal gas constant
$x = k / V_g (T + \Theta)^a$
$k = K \Sigma_j X_i k_i$

with empirical constants a, b, K, Θ, and k_i. The constants k_i, one for each molecular species, are covolumes. The covolumes are multiplied by their mole fraction of species X_i and are added to find an effective volume for a mixture. For a particular explosive, if we know the composition of detonation products, a, b, Θ, K, and all k_is can be found in Ref. 15.

The internal energy is determined by thermodynamics relation

$$\left(\frac{\partial E_g}{\partial V_g}\right)_T = T_g\left(\frac{\partial p_g}{\partial T_g}\right)_V - p_g \tag{19}$$

Integration of this equation for a fixed composition of the detonation products will allow us to calculate the energy of the detonation products as a function of temperature and volume. For each component, its thermodynamic properties as functions of temperature were calculated from the NASA tables compiled by Gordon and McBride.[16]

The BKW equation-of-state is the most commonly used and well-calibrated of those equations-of-state used to calculate the properties of detonation products. The detailed discussion and review of the BKW equation-of-state can be found in Ref. 15.

Numerical Method of Solutions

The system of partial differential equations described in the previous paragraph is integrated numerically. The Second Order Godunov method is used for the integration of the subsystem of equations describing flow of gaseous phase material and is described in Ref. 17. In the following, we will elaborate only on some specifics of its application to simulations of detonation products. The subsystem of equations describing the flow of particles is integrated using a simple upwind integration. This is done because our mathematical model neglects the pressure of interparticle interaction, and that prevents formulation of a Second Order Godunov scheme for particles.

The physical system under study will have concentrations of solid explosive powder ranging from 1000 kg/m^3 near the

ground to 0.75 kg/m^3 or less in the cloud. Detonation of this mixture will create detonation products with effective γ ranging from 3 to 1.1. To describe the flow of detonation products, we use the BKW equation-of-state described above. Since the Second Order Godunov method uses primitive variables to calculate Riemann problems at the edges of the cells, its implementation for non-ideal EOS is difficult. In our simulations, we have resolved this problem by using direct and inverse equations-of-state. After integrating a system of gas conservation laws, we use the direct BKW equation-of-state to calculate pressure, gamma, and temperature as functions of thermal energy, density, and mixture composition. After this step, we have a complete set of parameters allowing calculation of the fluxes in the Second Order Godunov method as well as interaction of the multiphase processes. The "inverse" EOS calculates internal energy as a function of density, pressure, and mixture composition. In our code, we use the "inverse" EOS to calculate the fluxes of conserved variables after calculation of the flux of primitive variables.

For the multiphase system under study, dx=dy=1mm was used to allow explicit integration of the gasdynamic and physical processes of evaporation and heat release. When a mismatch occurred between the physical and gasdynamical characteristic times, the time step was adjusted by some fraction to assure stability. However, this did not result in a significantly smaller time step than the one calculated using CFL criteria. For larger cell sizes, this approach will be impractical. Recently, we implemented a scheme in which multiphase processes are calculated implicitly; however, this will be reported elsewhere.

The numerical method is implemented in a code named MPHASE, which is fully vectorized and supported by number of graphics and diagnostics codes.

Results

Model Validation for One-Dimensional Detonation Wave Problem

The main advantage of our particle combustion model is its description of the phenomenology of detonation for a wide

Table 1 One-dimensional validation result

D(m/sec) - Detonation wave velocity,
P_{CJ}(Pa) - Pressure at Chapman-Jouguet Point
P_p(Pa) - Peak pressure; ρ_p(kg/m^3) - Peak density

RDX density (kg/m^3)	Parameters	Present calculation	Expt'l Ref. 1	Tiger calculation - Ref. 2	BKW calculation Ref. 1	Soviet experiments Ref. 3
1000 kg/m^3	D	6156	5981		6128	
	P_{CJ}	1.220×10^{10}			1.06×10^{10}	1.06×10^{10}
	P_p	2.57×10^{10}				
	ρ_p	1926				
800 kg/m^3	D	6051		6000		
	P_{CJ}	0.906×10^{10}		0.80×10^{10}		0.82×10^{10}
	P_p	1.86×10^{10}				
	ρ_p	1722				
400 kg/m^3	D	4000		4000		
	P_{CJ}	0.370×10^{10}		0.26×10^{10}	0.3×10^{10}	
	P_p	0.625×10^{10}				
	ρ_p	924				
250 kg/m^3	D	4040		3600		
	P_{CJ}	0.2476×10^{10}		0.13×10^{10}		
	P_p	0.4530×10^{10}				
	ρ_p	552				
100 kg/m^3	D	3496				
	P_{CJ}	0.5033×10^{9}				
	P_p	0.7650×10^{9}				
	ρ_p	720				
0.75 kg/m^3	D	1632	1410*	1670*		
	P_{CJ}	0.25×10^{7}	0.200×10^{7}*	0.26×10^{7}*		
	P_p	0.404×10^{7}				
	ρ_p	8				

Ref. 1 Mader, C., Numerical Modeling of Detonations, (University of California Press, Ltd., 1979) p. 47. Ref. 2 - Windermann, A., "An Evaluation of Unmodel Layer Loading Effects," IITRI Report, Feb. 1980. Ref. 3 - Stanyukovich, K. P., "Physics of Explosion" (in Russian), Nauka, 1975.

range of explosive particle sizes and densities. We will demonstrate this capability on a set of one-dimensional test problems. For these test problems, we simulated the initiation and propagation of the detonation waves in a shock tube-like setting, where the explosive particles are distributed uniformly through the shock tube volume.

Results of these simulations are summarized in Table 1, which shows detonation wave velocity, peak pressure, and peak density given as a function of the average density of the solid explosive. Here, the explosive two-phase mixture is composed from RDX particle and air, where RDX particle concentration varies from 0.75 kg/m^3 to 1000 kg/m^3. This concentration variation covers a whole range of solid explosive concentrations of interest to our problem. The simulations performed with the MPHASE code were compared with the experimental results[15,18] and calculations done with the TIGER code that are presented in Ref. 19.

From Table 1, it is clear that our simulation results compare favorably with other simulation results and experimental data. The maximum deviation between our results and referenced results is no greater than 15% for the entire range of explosives densities. Considering that our results were obtained with a single model for particle combustion applied to the extreme range of densities, our model gives an excellent prediction of the detonation wave parameters.

Two-Dimensional Simulation Results

Figure 1 shows a setup for a typical simulation with a computational domain of 25 cm × 25 cm. The explosive powder density is distributed according to the 4th power law of vertical distance, starting from the ground where the density is 1000 kg/m^3, to 1.2 cm, where the density is 0.75 kg/m^3. From this point to 25 cm height, the density is constant and equal to 0.75 kg/m^3. The density distribution in the direction of the "x" axis is uniform. The boundary conditions for the computational domain shown in Fig. 1 are specified as follows: solid wall along the "x" axis, symmetry conditions along the "y" axis, supersonic outflow for upper boundary, and at the

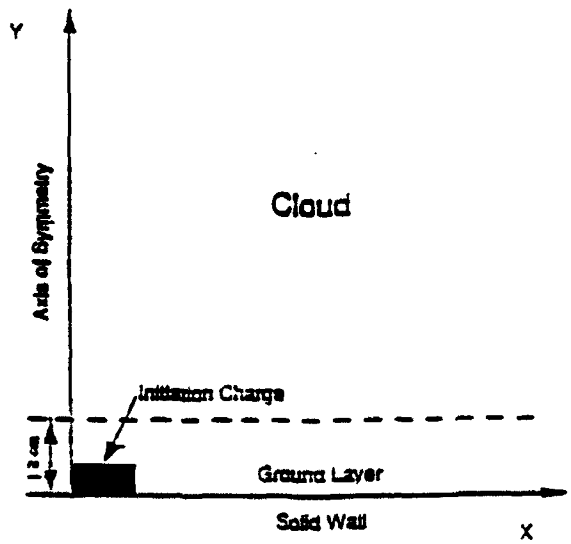


Fig. 1 Computational domain and boundary conditions.

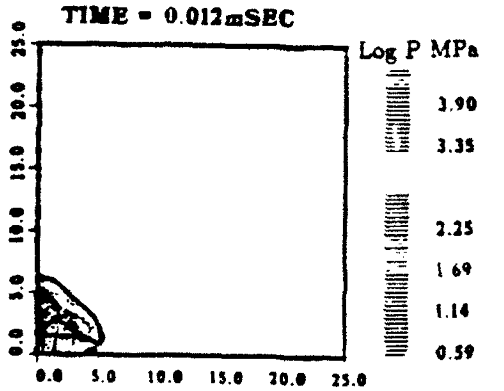
right of the computational domain. The mixture consists of RDX powder and air at ambient conditions, and it is assumed to be quiescent at the time of initiation.

The simulation starts at t=0 when the mixture is initiated at the lower left corner of the computational domain, as shown in Fig. 1. The energy released by the initiating explosion leads to formation of the detonation wave propagating through the multiphase media. Figure 2a shows pressure contours for the propagating detonation wave at the time of t=0.012 msec after initiation. The pressure contour levels are shown on the logarithmic scale in MPa. The maximum pressure value of 7940 MPa is observed in the layer of condensed explosive located near the ground. The pressure in the layer is two to three orders of magnitude higher than pressure behind the detonation wave in the 0.75 kg/m^3 RDX cloud and air located above the distance of 1.2 cm from the ground. Figure 2a demonstrates that the detonation wave in the cloud is overdriven, since the pressure behind the shock continuously rises and reaches its maximum in the layer. From this figure, we also observe that the overdriven wave propagates faster in the cloud than in the layer. This is explained by the fact that it is easier to compress air that is very lightly loaded with particles and located above the ground layer than it is to compress air heavily loaded with a particle mixture near the ground. It is interesting to note a discontinuous pressure change between the yellow contours and the light blue and green contours behind the detonation front. This discontinuity is overemphasized by our presentation of contour lines on the logarithmic scale; however, further examination of our simulation results indicates this feature is real and is similar in nature to barrel shocks observed for strong jets.

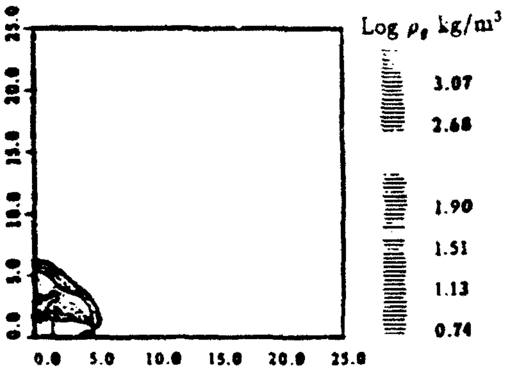
In Fig. 2b, gas-phase density contours are shown for the time t=0.012 msec. Here the contour lines are distributed on the logarithmic scale. The main features of the shock wave structure are very similar to those observed in the pressure contours figure. We see that a jet of high-density gases reflects from the center of symmetry axis. which will create a contact discontinuity that we will observe at later times. The barrel

shock is clearly visible in this figure. In Fig. 2c, the particle density contour plots are shown for t=0.012 msec. The contour levels in Fig. 2c are given on the logarithmic scale and the initial deposition of the explosive material in the ground layer of the computational domain can be clearly observed. The white contour line delineates the beginning and the end of the reaction zone in the cloud. To the left of these contours lies an area with combustion products and to the right are unburned particles in the cloud. The reaction zone length is of the order of 1 cm.

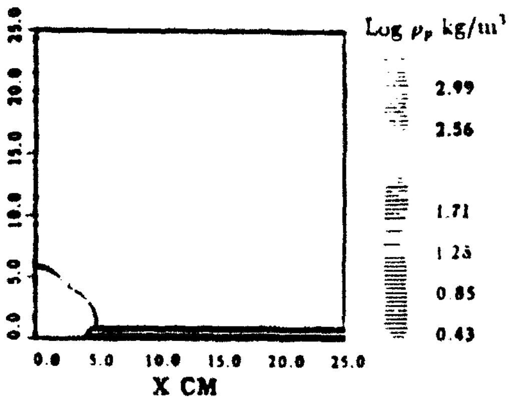
Figure 2d shows pressure contours for the same simulation for the time t=0.055 msec, just before the detonation wave leaves the computational domain. In this figure, we see that the global structure of the wave did change slightly from Fig. 2a. We observe that the barrel shock wave is fully developed and has a half-ellipse shape. The detonation wave in the cloud is still overdriven; however, part of the shock wave front that propagates vertically weakened because it gets further away from the detonation front in the layer. Another noticeable feature is the increase in distance between the detonation front in the layer and in the cloud area close to the layer. This is a result of the fact that the lightly loaded two-phase media above the layer can be compressed much more easily than the particle-heavy ground layer. In Fig. 2e, temperature contours are shown for t=0.055 msec. Comparing this figure with an early stage of the wave propagation, we observe a significant cooling of the front area propagating upwards, which indicates transition from the overdriven detonation regime to a self-sustained detonation. We also observe in Fig. 2a clear development of two detonation fronts, one moving vertically in the cloud and another moving horizontally in the layer. Because the energy density of the explosive powder in the layer is about three orders of magnitude larger than in the cloud, the vertical parts of the front represent an overdriven detonation wave in the cloud. Even though the vertical front has slowed down compared with the horizontal front, its speed and parameters far exceed those typical for detonation waves in a cloud. In fact, the self-sustained detonation regime in the cloud will



a) Pressure.

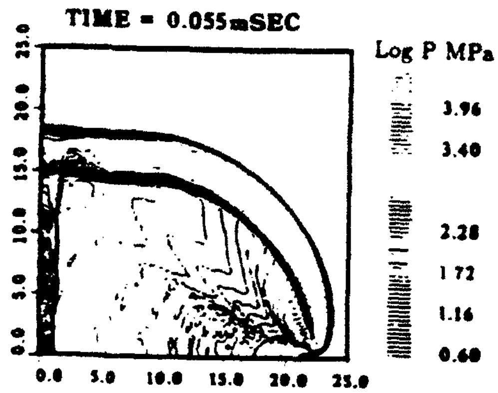


b) Gas Density.

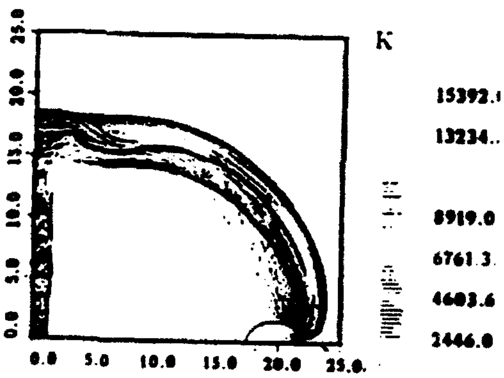


c) Particle Density.

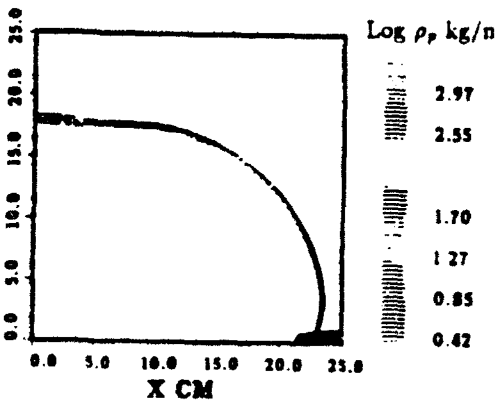
Fig. 2 Fourth power layer distribution: maximum density in the layer 800 kg/m^3; density in the cloud 0.75 kg/m^3; time 0.012 m/s and 0.055 m/s after initiation.



d) Pressure.



e) Temperature.



f) Particle Density.

Fig. 2 (continued) Fourth power layer distribution: maximum density in the layer 800 kg/m^3; density in the cloud 0.75 kg/m^3; time 0.012 m/s and 0.055 m/s after initiation.

develop at the distance of about 3 m from the layer. The area of the front close to the detonation wave in the layer will remain hot and overdriven, since it is located very close to the detonation front in the layer. In Fig. 2f, particle density contours are shown on a logarithmic scale. We can clearly observe the reaction zone delineated by black contour lines. In this case, the reaction zone length in the cloud is about 1 cm. Consistent with the gradual transition from overdriven to self-sustained detonation, the reaction zone length is larger for the vertical part of the detonation front. The detonation wave velocity observed in our simulation is approximately 4048 msec, which is significantly lower than the detonation wave velocity observed in RDX with a density of 860 kg/m^3 (see Table 1), the highest density in the ground layer. This can be explained by high gradient of particle density distribution in the layer, where the density drops rapidly from 860 kg/m^3 at the bottom of the layer to 1 kg/m^3 at the top strata of the layer at 12 mm above the ground.

To further explore properties and phenomenology of the detonation waves propagating in the layer/cloud systems, we simulated additional cases in which explosive powder density distribution was different from the case reported above, although total weight of fuel per unit area remained the same.

In Fig. 3, results are shown for the case of a uniform 2.5 cm-thick layer of RDX with a density of 100 kg/m^3 and a 0.75 kg/m^3 cloud initiated under the same conditions as in the previous example. Figures 3a, 3b, and 3c show pressure, gas density, and particle density contour plots at t=0.066 msec. We observe that because the layer has considerably smaller density compared to the case reported above, the precursor effect of the detonation wave in the cloud preceding the wave in the layer is less pronounced. Also, one can observe a significant difference in the shape of the strong contact discontinuity in the region of the shock front close to the layer. In Fig. 3b, we can clearly distinguish two contact surfaces. One is between condensed explosive detonation products in the layer and in the cloud, and another is between the detonation products from

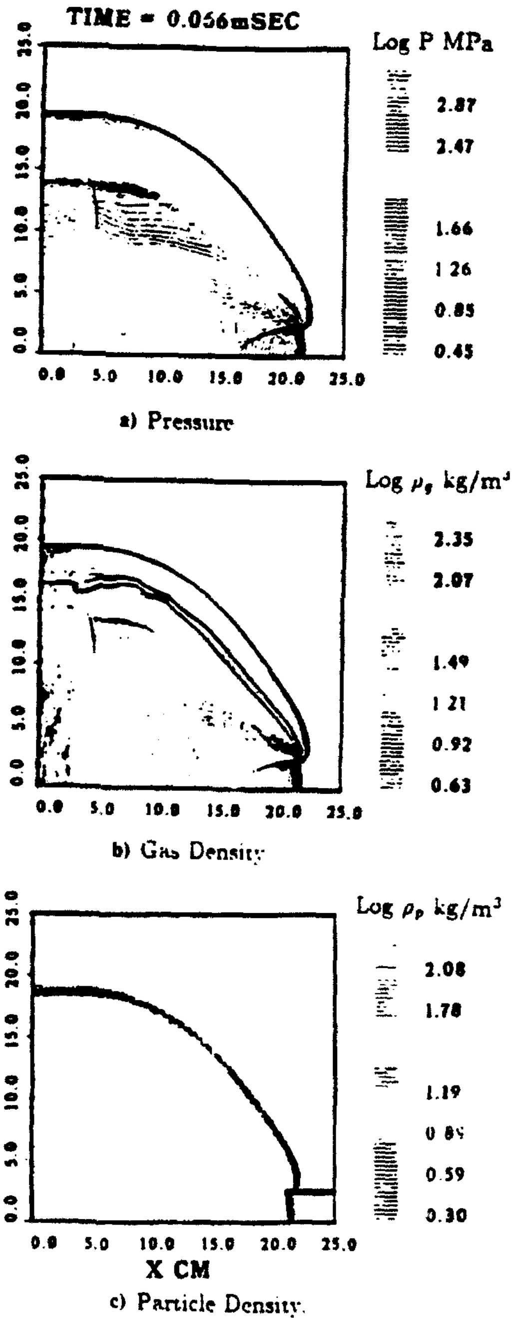
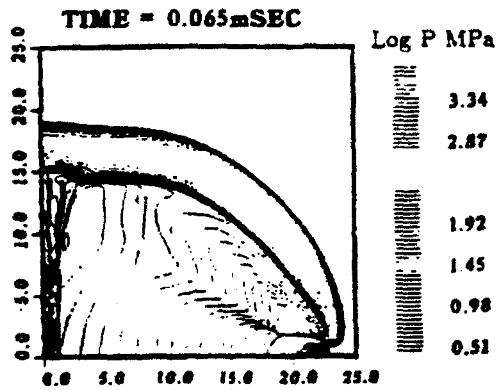
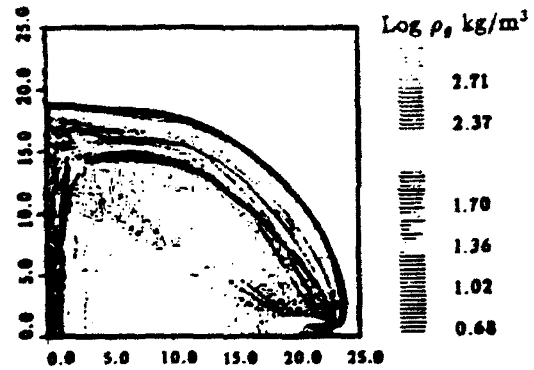


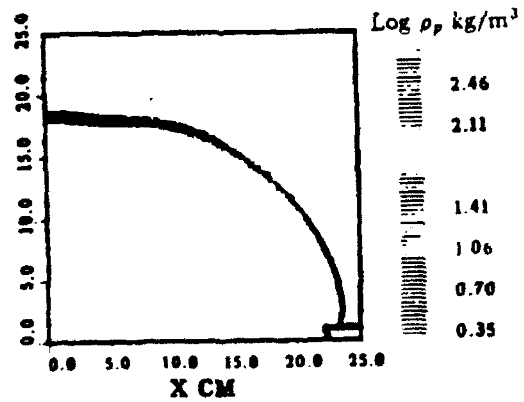
Fig. 3 Constant density 2.5-cm-thick layer; maximum density in the layer 100 kg/m^3; density in the cloud 0.75 kg/m^3; time 0.055 m/s after initiation.



a) Pressure.



b) Gas Density.



c) Particle Density.

Fig. 4 Constant density 1.2-cm-thick layer; maximum density in the layer 250 kg/m^3; density in the cloud 0.75 kg/m^3.

layer explosive detonation and from cloud particle detonation. We should note that these contact surfaces are overemphasized by the logarithmic display of the contour plot levels. The maximum pressure observed in this simulation is 955 MPa, which is about one order of magnitude smaller than in previous simulation. This is consistent with the one order of magnitude difference in the maximum density of the ground layer in the two cases. The detonation wave speed in the case presented in Fig. 3 is 3407 msec. That is only slightly lower than the speed predicted by one-dimensional simulations presented in Table 1, which reflects thc influence of the two-dimensional expansion on the detonation wave propagation.

Figure 4 presents results for the case of a uniform density of 250 kg/m^3 in 1.2 cm ground layer. All other parameters are the same as in the previous two cases. In Figs. 4a, 4b, and 4c, pressure, gas density, and particle density contour plots are shown at the time t=0.066 msec after initiation of the detonation wave. Here, the detonation wave propagates faster than in the previous cases U=3660 msec. This is about 400 msec slower than in the case of parabolic density distribution. Maximum pressure on the ground is 2150 MPa, which is consistent with the increase of powder density in the layer. The basic structure of the detonation front and the contact surfaces is similar to the case of parabolic density distribution.

Conclusions

We have presented a mathematical model and numerical solution for the simulation of initiation and propagation of the detonation waves in multiphase mixtures consisting of solid combustible particles and gas. Using this model, we studied detonations in mixtures of solid RDX particles and air, with the objective of examining the effects of wide variation in particle density distribution on the dynamics and structure of detonation waves. We considered a physical system of solid particle clouds in air, in which a significant amount of particles settle on the ground and the condensed-phase concentrations in the particle/air mixture range from 0 to 1000 kg/m^3. This range of solid-phase densities necessitated development of the

model and its numerical implementation for a wide range of particle concentrations. Our validation study has shown good agreement between the simulations and referenced results for the whole range of particle concentrations.

Two-dimensional simulations were done for the system of low particle density concentration clouds and ground layers formed by high concentrations of the RDX powder. We examined three cases of ground layer density distribution: a fourth power distribution within 12 mm above ground with a maximum density on the ground of 860 kg/m^3; a uniform 25 mm-thick layer with a density of 100 kg/m^3; and a 12 mm-thick uniform layer with a density of 250 kg/m^3. In all these cases, the weight of condensed phase per unit area was the same, which allowed examination of the effects of the particle density distribution on detonation wave parameters.

In all examined two-dimensional cases, the detonation wave in the cloud in the computational domain was significantly overdriven and did not play an important role. We estimated that the self-sustained regime of the detonation wave in the cloud for the examined cloud concentrations can occur only at the distances of 2–3 m above ground. At the same time, the particle density distribution in the layer determines the dynamics of the detonation wave as well as pressure on the ground.

In all three two-dimensional simulations, we observed a very distinct shape of the detonation wave front in the vicinity of the layer. In this area, the overdriven detonation in the cloud is preceding the detonation wave in the ground layer. This feature of the detonation front can be explained by the fact that the energy released in the detonation wave in the ground layer produces a faster shock wave in the dilute cloud than in those heavily loaded with solid particle stratas from the ground layer. However, these structures were not observed experimentally, and more studies are needed to examine their parameters.

The maximum pressure affecting the ground was directly related to the maximum particle density in the lower strata of the layer. However, the detonation front velocity for the fourth

power distribution case was considerably lower than calculated for a one-dimensional case with 860 kg/m^3 particle density, reflecting the significant effect of two-dimensional expansion. Two other cases with 250 kg/m^3 and 100 kg/m^3 maximum densities had the detonation wave velocity only slightly lower than the one-dimensional simulations of the same RDX/air concentrations. It is interesting to compare the simulation of the fourth power density distribution case and 250 kg/m^3 case. In both cases, the same amount of explosive was distributed in the same physical space; however, the parameters of developed detonations were vastly different. Existence of the high-density strata at the bottom of the ground layer in the fourth power case significantly increased the maximum pressure at the ground and produced higher detonation wave velocity.

Using a variable density layer, one can reach a combination of pressure and velocity conditions outside of Chapmen-Jougett limitations. The range of conditions that can be obtained in the variable density system and the parametrics for this range need a more systematic study. In this article, we introduced only the mathematical formulation and numerical simulation method validated for the range of conditions of interest. In addition, we have given some examples of its application for two-dimensional simulations. However, this methodology should be linked to an experimental study for a more in-depth analysis of the phenomenology discussed here.

References

[1] Eidelman, S., Timnat, Y. M., and Burcat, A., "The Problem of a Strong Point Explosion in a Combustible Medium," 6th Symposium on Detonation, Office of Naval Research, Coronado, CA, 1976, p. 590.

[2] Burcat, A., Eidelman, S., and Manheimer-Timnat, Y., "The Evolution of a Shock Wave Generated by a Point Explosion in a Combustible Medium," Symposium of High Dynamic Pressures (H.D.P.), Paris, 1978, p. 347.

[3] Oved, Y., Eidelman, S., and Burcat, A., "The Propagation of Blasts from Solid Explosives to Two-Phase Medium," *Propellants and Explosives*, Vol. 3, No. 105, 1978.

[4] Eidelman, S., and Burcat, A., "The Evolution of a Detonation Wave in a Cloud of Fuel Droplets; Part I, Influence of the Igniting Explosion," *AIAA Journal*, Vol. 18, 1980, p. 1103.

[5]Liu, J. C., Kauffman, C. W., and Sichel, M., "The Lateral Interaction of Detonating and Detonable Mixtures," Private communication, 1990.

[6]Kuo, K., *Principles of Combustion*, John Wiley and Sons, Inc., New York, NY, 1990, pp. 513-626.

[7]Kauffman, C. W., et al., "Shock Wave Initiated Combustion of Grain Dust," *Symposium on Grain Dust*, Manhattan, KS, 1979.

[8]Eidelman, S., and Burcat, A., "Numerical Solution of a Non-Steady Blast Wave Propagation in Two-Phase ('Separated Flow') Reactive Medium," *Journal of Computational Physics*, Vol. 39, 1981, p. 456.

[9]Reinecke, W. G., and Waldman, G. D., "Shock Layer Shattering of Cloud Drops in Reentry Flight," AIAA Paper 75-152, 1975.

[10]Eidelman, S., and Burcat, A., "The Mechanism of Detonation Wave Enhancement in a Two-Phase Combustible Medium," 18th Symposium on Combustion, The Combustion Institute, Waterloo, Ontario, Canada, 1980, pp. 1661-1670.

[11]*Engineering Design Handbook, Explosives Series, Properties of Explosives of Military Interest*, AMC Pamphlet, AMCP 706-7177, 1971.

[12]Drake, R. M., Jr., "Discussions on G. C. Vliet and G. Leppert: Forced Convection Heat Transfer from an Isothermal Sphere to Water," *Journal of Heat Transfer*, Vol. 83, 1961, p. 170.

[13]Schlichting, H., *Boundary Layer Theory* 7th ed., McGraw-Hill, New York, 1983.

[14]Cowan, R. D., and Fickett, W., "Calculation of the Detonation Products of Solid Explosives with the Kistiakowsky-Wilson Equation of State," *Journal of Chemical Physics*, Vol. 24, 1956, p. 932.

[15]Mader, C. L., *Numerical Modeling of Detonation*, University of California Press, Ltd. London, England, 1979.

[16]Gordon, S., and McBride, B. J., "Computer Program for Calculations of Complex Chemical Equilibrium Compositions, Rocket Performance, Incident and Reflected Shocks and C-J Detonations," NASA SP-273, 1976 (revision).

[17]Eidelman, S., Collela, P., and Shreeve, R. P., "Application of the Godunov Method and Its Second Order Extension to Cascade Flow Modelling," *AIAA Journal*, Vol. 22, 1984, p. 10.

[18]Stanukovitch, K. P., *Physics of Explosion* (in Russian), Nauka, 1975.

[19]Wiedermann, A., "An Evaluation of Bimodal Layer Loading Effects," *IITRI Report*, February 1990.

Combust. Sci. and Tech., 1993, Vol. 89, pp. 201-218
Photocopying permitted by license only

Detonation Wave Propagation in Combustible Particle/Air Mixture with Variable Particle Density Distributions

SHMUEL EIDELMAN and XIAOLONG YANG *Science Applications International Corporation McLean, VA 22102*

(Received June 11, 1991; in final form June 18 1992)

Abstract—A mathematical model is presented describing a physical system of detonation waves propagating in a solid particle/air mixture with a wide range of solid phase concentrations. The mathematical model was solved numerically using the Second Order Godunov method, and numerical solutions were validated for detonation waves propagating in mixtures with concentrations of solid phase from 0.75 kg/m^3 to 1000 kg/m^3. Numerical solution was obtained for detonation waves propagating in a system consisting of clouds with a small concentration of particles and a ground layer in which solid particle densities are three orders of magnitude larger than in the cloud. Three different particle concentration distributions in the ground layer were simulated and compared in terms of detonation wave structure and parameters.

Key words. detonation wave, two-phase flow, numerical simulation

1. INTRODUCTION

When combustible particles are intentionally or unintentionally dispersed into the air, the resulting mixture can be detonable. Formation of this potentially explosive dust environment and the properties of its detonation are of significant practical interest in view of its destructive or creative effects. The experimental and theoretical study of these phenomena until now has addressed only homogenous particle/oxidizer mixtures. However, intentional or accidental processes of the explosive dust dispersion will always lead to inhomogeneous particle density distribution. Some industrial methods of explosive forming rely on detonation of explosive powder. This powder can be deposited as a thin layer over the surface area of the forming metal, with some remaining concentration in the vicinity of the layer. The structure of the detonation waves and the phenomenology of their initiation and propagation in these environments are the main subjects of this paper.

When the detonation wave is generated in a homogeneous mixture by a "direct initiation," it starts with a strong blast wave from the initiating charge. As the blast wave decays, combustion of the reactive mixture behind its shock front starts to have a larger role in support of the shock wave motion. When the initial explosion energy exceeds some critical value, transition to steady state detonation occurs (cf. Eidelman *et al.*, 1976; Burcat *et al.*, 1978; Oved *et al.*, 1978; Eidelman and Burcat, 1980). In explosive dust mixtures with a nonuniform distribution of particle density, the initiation dynamics is significantly more complicated. The critical initiation energy sufficient for one of the explosive particle density strata regions is not necessarily adequate for other regions. Also, when there is a significant variation in density between the different layers (regions) of the mixture, steady detonation in one layer can result in an overdriven detonation in an adjacent layer. Our paper demonstrates that the phenomenology of these interactions is distinctly different from the classical studies of multi-layer detonations in gases. This is primarily because the energy content of adjacent layers in a typical multi-gas layer experiment varies by a factor of two or four (Liu *et al.*, 1990), whereas the energy content in explosive dust/air mixtures can vary by several orders of magnitude.

In this paper we use detailed numerical simulation to study the initiation dynamics and propagation phenomenology for a general case of explosive dust dispersion. We will consider particle density variation from 1000 kg/m^3 in the ground layer to 0.75 kg/m^3 for the upper edges of the cloud. The effects of the cloud density variation on detonation wave parameters will be examined for different cases of cloud particle density distribution. When possible, the results of computer simulations are validated in comparison with experimental and theoretical studies.

The outline of this paper is as follows. Section 2 gives a description of mathematical model that includes governing conservation equations for two phases and the constitutive laws. We describe the model for a particle gas interaction, combustion and equation-of-state for gas phase. The numerical integration technique for solving the mathematical model will also be outlined. In Section 3, we present our numerical simulation results. We first validate our model by comparing one dimensional detonation wave simulation with available experimental results. We then give the two dimensional simulation for detonation wave propagation in combustible particles/air mixtures with variable particle density distribution. Concluding remarks are given in Section 4.

2. THE MATHEMATICAL MODEL AND THE NUMERICAL SOLUTION

The mathematical model consists of conservation governing equations and constitutive laws that provide closure relations for the model. The basic formulation adopted here follows the two-phase fluid dynamics model presented in the text by Kuo (1990). The approach assumes that there are two distinct continua, one for gas and one for solid particles, each moving at its own velocity through its own control volume. The sum of these two volumes represents an average mixture volume. Furthermore, particles in their own control volume are assumed monodisperse and they are moving with the same velocity. With these assumptions, distinct equations for continuity, momentum and energy are written for each phase. The interaction effects between the two phases are accounted as the source terms on the right hand side of the governing equation. The following is a short description of the two phase flow model used in our study, with conservation equations written in Eulerian form for two dimensional flow in Cartesian coordinates.

Conservation Equations

Continuity of gaseous phase:

$$\frac{\partial \rho_1}{\partial t} + \frac{\partial(\rho_1 u_g)}{\partial x} + \frac{\partial(\rho_1 v_g)}{\partial y} = \Gamma; \tag{2.1}$$

Continuity of solid particle phase:

$$\frac{\partial \rho_2}{\partial t} + \frac{\partial(\rho_2 u_p)}{\partial x} + \frac{\partial(\rho_2 v_p)}{\partial y} = -\Gamma; \tag{2.2}$$

Conservation of momentum of gaseous phase in x-direction:

$$\frac{\partial(\rho_1 u_g)}{\partial t} + \frac{\partial(\rho_1 u_g^2 + \phi p_g)}{\partial x} + \frac{\partial(\rho_1 u_g v_g)}{\partial y} = -F_x + \Gamma u_p; \tag{2.3}$$

Conservation of momentum of solid particle phase in y-direction:

$$\frac{\partial(\rho_1 v_g)}{\partial t} + \frac{\partial(\rho_1 u_g v_g)}{\partial x} + \frac{\partial(\rho_1 v_g^2 + \phi p_g)}{\partial y} = -F_y + \Gamma v_p; \tag{2.4}$$

Conservation of momentum of solid particle phase in x-direction:

$$\frac{\partial(\rho_2 u_p)}{\partial t} + \frac{\partial(\rho_2 u_p^2)}{\partial x} + \frac{\partial(\rho_2 v_p u_p)}{\partial y} = F_x - \Gamma u_p; \tag{2.5}$$

Conservation of momentum of solid particle phase in y-direction:

$$\frac{\partial(\rho_2 v_p)}{\partial t} + \frac{\partial(\rho_2 u_p v_p)}{\partial x} + \frac{\partial(\rho_2 v_p^2)}{\partial y} = F_y - \Gamma v_p; \tag{2.6}$$

Conservation of energy of gas phase:

$$\frac{\partial(\rho_1 E_{gT}}{\partial t} + \frac{\partial(\rho_1 u_g E_{gT} + u_g \phi p_g)}{\partial x} + \frac{\partial(\rho_1 v_g E_{gT} + v_g \phi p_g)}{\partial y} =$$

$$\Gamma\left(\frac{u_p^2 + v_p^2}{2} + E_{chem} + C_s \bar{T}_p\right) - \left(F_x u_p + F_y v_p\right) = \dot{Q}; \tag{2.7}$$

Conservation of energy of solid particle phase:

$$\frac{\partial(\rho_2 E_{pT})}{\partial t} + \frac{\partial(\rho_2 E_{pT} u_p)}{\partial x} + \frac{\partial}{\partial y}(\rho_2 E_{pt} v p = \dot{Q} + (F_x v_p + F_y v_p)$$

$$-\Gamma\left(\frac{u_p^2 + v_p^2}{2} + E_{chem} + C_s \bar{T}_p\right); \tag{2.8}$$

Conservation of number density of solid particle:

$$\frac{\partial N_p}{\partial T} + \frac{\partial(N_p u_p)}{\partial x} + \frac{\partial(N_p v_p)}{\partial y} = 0. \tag{2.9}$$

In the above equations, $\phi = 1 - \frac{N_p M_p}{\rho_p}, \rho_1 = \phi\rho_g, \rho_2 = (1-\phi)\rho_p$, where N_p and M_p are the number density of particles and mass of each particle, respectively, and ρ_g and ρ_p are the material density of gas and particle densities, respectively. u_g, v_g, p_g are gas phase x-velocity, y-velocity and pressure, respectively; u_p, v_p, $\bar{T}_p$, are x-velocity, y-velocity and average particle temperature, respectively. C_s is the solid particle specific heat, and $E_{chem} = E_{comb} - E_{evap}$, where E_{comb} is heat of combustion and E_{evap} is heat of evaporation. Γ is the rate of phase change from solid to gas and Q is heat transfer between the two phases: F_x, F_y are drag force between the two phases in x and y directions, respectively.

Equations (2.2) and (2.9) are linked through the relation $\rho_2 = N_p M_p$. In the case of a reactive solid phase, M_p decreases due to combustion. The mass of a single particle at any point can be obtained from $M_p = \rho_2(x,y)/N_p(x,y)$, and the diameter of a particle at any spatial location is $D(x,y) = [6M_p(x,y)/\pi\rho_p]1/3$. The total internal energy of gaseous phase

$$E_{gT} = E_g + \frac{1}{2}(u_g^2 + v_g^2) \text{ and } E_g = E_g(p_g, \rho_g) \tag{2.10}$$

where $E_g(p_g, \rho_g)$ is the equation-of-state for gas phase, which will be discussed later.

The total internal energy of solid particle phase is

$$E_{pT} = E_p + \frac{1}{2}(u_p^2 + v_p^2) \quad \text{and} \quad E_p = E_{comb} + C_s\bar{T}_p. \qquad (2.1)$$

In order to close the above system of conservation equations, it is necessary to define certain criteria and interaction laws between the two phases, which include mass generation rate, Γ, drag force between particles and gas, F_x, F_y and the interphase heat transfer rate $\dot{Q}$. The model for particle and gas interaction and particle combustion that results in the constitutive relation for the conservation equations, is explained in detail in the next subsection.

Model for a Particle Gas Interaction and Combustion

Presently the physics of the energy release mechanisms in solid particles/air mixtures is not clearly understood. This can be attributed to the obvious difficulties of making a direct non-obtrusive measurement in the optically thick environment typical for this system. In the experimental and theoretical work done for the grain dust detonation conditions (Kauffman *et al.*, (1979), it was demonstrated that the volatile components released by the particle heated behind the shock front play a major role in determining the detonability limits of the mixture. Eidelman and Burcat (1981) successfully applied a combination of fast evaporation and aerodynamic shattering mechanisms to simulate a two-phase detonation process.

The chemical processes of a single particle combustion, which mainly occur in the gaseous phase, are significantly faster than the physical processes of particle gasification or disintegration. Thus, in the multi-phase mixtures, the rate of energy release will be mostly determined by physics of particle disintegration. It is very difficult to describe the details of particle disintegration in the complex environment prevalent behind the shock or detonation wave. For example, Reinecke and Waldman (1975) defined five different disintegration regimes for a relatively simple environment of water droplets passing through a weak shock. Fortunately, in most cases of multi-phase detonation, only the main features of the particle disintegration dynamics need to be captured to describe the phenomena. For example, Eidelman and Burcat (1980) used simple models for particle evaporation and shattering to obtain simulation results that compared very favorably with experimental data. Because of our inability to resolve the particle disintegration problem in all its complexity, the validation of the model against known experimental data is essential.

In this paper we consider solid particles consisting of explosive material. Explosive material contains fuel and oxidizer in a passive state at low temperature; however, when the temperature rises the fuel and oxidizer react, leading to detonation or combustion. The initiation for explosives will occur at a relatively low temperature. For example, TNT will detonate when heated to the temperature of 570°C. Only particles larger than a critical detonation size can detonate directly when initiated by a shock wave. We consider here particles smaller than 4mm in diameter that will not detonate when heated, but will burn when the temperature on the particle surface reaches a critical value. Since the heat conduction inside the explosive material is relatively slow, the process of particle heating needs to be resolved in detail. Our simulations numerically solve the temperature field in the particles at every time step of numerical integration of the global conservation equations. The explosive particle combustion model examined in this paper assumes that the fraction of the particle that reaches the critical temperature will burn instantaneously. Energy transfer by convection and conduction is simulated by solving the unsteady heat

conduction equation in each computational cell at each time step. Assuming a particle's temperature T_p to be a function of time and radial position only, the unsteady heat conduction equation may be transformed to:

$$\frac{d^2w}{dr^2} = \frac{1}{\alpha}\frac{dw}{dt}, \tag{2.12}$$

subject to the boundary conditions:

$$w = 0 \; at \; r = 0, \; t > 0$$

$$k\frac{dw}{dr} = (h - \frac{1}{R})w = hRT_g \; at \; r = R, t > 0 \tag{2.13}$$

where:

$w(r,t)$ = $rT_p(r,t)$
r = radial position
T(r,t) = temperature
R = partial radius
T_g = temperature of surrounding gas
k = thermal conductivity of particle
h = convective heat transfer coefficient.

The Nusselt number, used to find h, is given by an empirical relation given by Drake (1961). The gas viscosity is found from Sutherland's Law. The gas thermal conductivity is calculated by assuming a constant Prandtl number. Lastly, the boiling temperature at a given pressure is found from the Clapeyron-Clausius equation under the assumptions of: 1) constant latent enthalpy of phase change, 2) the vapor obeys the ideal equation-of-state, and 3) the specific volume of the solid/liquid is negligible compared to that of the vapor. A critical temperature is also employed to serve as an upper limit to the boiling point, regardless of pressure.

Equation (2.12) with boundary condition (2.13) can be numerically integrated using either implicit or explicit schemes, which will be explained later.

Knowledge of the particle temperature profile also allows us to determine, Γ, the rate of phase change from solid particle to gas. Once any point at a radial location $0 \leq r \leq R$ has a temperature exceeding the boiling temperature, the entire mass between r and R is transferred to the gas phase in one time step. In so doing, an energy equal to the product of the mass lost and the particle combustion of heat minus heat of evaporation energy is transferred from the particle to the gas.

The interphase drag forces (Fx, Fy) are determined from the experimental drag for a sphere, as presented by Schlichting (1983).

$$F_x = \left(\frac{\pi}{8}\right) N_p \rho_g C_D |\mathrm{V_g} - \mathrm{V_p}|(\mathrm{u_g} - \mathrm{u_p})\mathrm{R}^2 \tag{2.14}$$

where

$$C_D = \begin{cases} \frac{24}{Re}\left(1 + \frac{Re^{2/3}}{6}\right) & for \; Re < 1000; \\ 0.44 & for \; Re > 1000. \end{cases} \tag{2.15}$$

and $Re = \frac{2R|V - V_p|}{\mu_g}$, R is the radius of the particle and μ_g is gas viscosity at a temperature of $T_{film} = \frac{1}{2}(T_g + \bar{T}_p)$. Similarly, the formula for F_y is

$$F_y = \frac{\pi}{8} N_p \rho_g C_D |\mathbf{v}_g - \mathbf{v}_p| (\mathbf{v}_g - \mathbf{v}_p) \mathbf{R}^2. \tag{2.16}$$

Equation of-State for Detonation Products

To close the system of governing equations, one needs a constitutive relation between density, pressure, temperature, and energy for gas phase, which is an equation-of-state. This study uses the Becker-Kistiakowsky-Wilson (BKW) equation-of-state (cf. Cowan and Fickett, 1956; Mader, 1979), which is,

$$p_g V_g / \bar{R} T_g = 1 + x e^{bx}. \tag{2.17}$$

where V_g = volume of gas phase
p_g = pressure of gas phase
T_g = temperature of gas phase
$\bar{R}$ = universal gas constant
x = $k / F_g (T + \Theta)^a$
k = $K \sum_i X_i k_i$

with empirical constants a, b, K, Θ and k_i. The constants k_i, one for each molecular species, are co-volumes. The co-volumes are multiplied by their mole fraction of species, X_i, and are added to find an effective volume for a mixture. For a particular explosive, if we know the composition of detonation products a, b, Θ, K, and all k_i's can be found in the book by Mader (1979).

The internal energy is determined by thermodynamics relation

$$\left(\frac{\partial E_g}{\partial V_g} \right)_T = T_g \left(\frac{\partial p_g}{\partial T_g} \right)_V - p_g. \tag{2.18}$$

Integration of this equation for a fixed composition of the detonation products will allow us to calculate the energy of the detonation products as a function of temperature and volume. The thermodynamic properties as functions of temperature were calculated for each component from the NASA tables compiled by Gordon and McBride (1976).

The BKW equation-of-state is the most used and well calibrated of those equations-of-state used to calculate the properties of detonation products. The detailed discussion and review of the BKW equation-of-state can be found in the literature (cf. Cowan and Fickett, 1956; Mader, 1979).

Numerical Method of Solutions

The system of partial differential equations described in the previous paragraph is integrated numerically. Equations (2.1)—(2.9) can be written in the following vector form

$$\frac{\partial \Phi}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = \Omega. \tag{2.19}$$

In order to numerically solve this equation, an operator time-splitting technique is used. Assuming that all flow variables are known at a given time, we can calculate its advancement in time by splitting the integration into two stages.

In the first stage, the conservative part of Eq. (2.19) is solved:

$$\frac{\partial \Phi}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = 0. \tag{2.20}$$

The Second Order Godunov method is used for the integration of the subsystem of equations describing the gaseous phase flow. The method is well documented in the literature (cf. Eidelman *et al.*, 1984; Colella, 1985; Colella and Glaz, 1985). In the following we will elaborate only some specifics of application of the method with BKW equation-of-state to simulate detonation product.

The physical system under study will have concentrations of solid explosive particle ranging from 1000 kg/m^3 near the ground to 0.75 kg/m^3 in the cloud. Detonation of this mixture will create detonation products with effective γ ranging from 3 to 1.1. To describe the flow of detonation products, we use the BKW equation-of-state described previously. Since the Second Order Godunov method uses primitive variables to calculate Riemann problems at the edges of the cells, its implementation for non-ideal EOS is difficult. In our simulations, we have resolved this problem by involving a local parameterization of EOS and by using direct and inverse equations-of-state. After integrating a system of gas conservation laws, we use the direct BKW equation-of-state to calculate pressure, gamma, and temperature as functions of thermal energy, density, and mixture composition. After this step, we have a complete set of parameters allowing calculation of the fluxes obtained from solving the Riemann problem (Colella and Glaz, 1985). The "inverse," EOS calculates internal energy as a function of density and pressure. In our code we use the "inverse" EOS to calculate the fluxes of conserved variables after calculation of the flux from Riemann problem of primitive variables.

The subsystem of equations describing the particle phase flow is integrated using a simple finite difference upwind scheme. This is done because there is no shock in the particle phase and the upwind scheme leads to a robust and accurate integration scheme.

In the second stage, the source term is added and the following equation is solved:

$$\frac{\partial \Phi}{\partial t} = \Omega. \tag{2.21}$$

To integrate this equation in time, we need to obtain Ω as a function of Φ. To do this, we first solve the particle heat conduction and heat transfer equation (2.12) with a boundary condition (2.13) that gives the temperature distribution as a function of particle radius and time using a local particle grid. Since the particle radius, R, will become very small due to evaporation, the implicit Crank-Nicolson algorithm is used because of its stability properties and its second order temporal and spatial accuracy. Using the Crank-Nicolson scheme to predict the particle temperature profiles at times t_1 and t_2 permits easy calculation of the total energy exchange, Q, between t_1 and t_2, due to convection and conduction. Knowing the temperature distribution inside the particle, we can calculate gas generation rate Γ, drag force F_x, F_y, and heat exchange Q, between two phases and hence, Ω of Eq. (2.21). After obtaining the source term, we can integrate Eq. (2.21) by an explicit scheme.

For the multiphase system under study, $\Delta_x = \Delta_y = 1mm$ was used to allow explicit integration of the gasdynamic and physical processes of evaporation and heat release. When a mismatch occurred between the physical and gasdynamical characteristic times, the time step was adjusted by some fraction to assure stability. However, the resulting time step was not significantly smaller than that calculated by CFL criteria. For larger cell sizes, this approach will be impractical.

The numerical method is implemented in a code named MPHASE, which is fully vectorized and supported by number of graphics and diagnostics codes.

Table I.
One Dimensional Validation Result

D[m/sec]—Detonation wave velocity,
P_{CJ}[Pa]—Pressure at Chapman-Jouguet Point
P_p[Pa]—Peak pressure: ρ_p[kg/m^3]—Peak density

RDX Density (kg/m^3	Parameters	Present Calculation	Expt'l Ref. 1	Tiger Calculation Ref. 2	BKW Calculation Ref. 1	Soviet Experiments Ref. 3
1000 kg/m^3	D	6155	5981		6128	
	P_{CJ}	1.220×10^{10}			1.08×10^{10}	1.00×10^{10}
	P_p	2.57×10^{10}				
	ρ_p	1936				
860 kg/m^3	D	6031		5900		
	P_{CJ}	0.986×10^{10}		0.88×10^{10}		0.82×10^{10}
	P_p	1.95×10^{10}				
	ρ_p	1722				
466 kg/m^3	D	4800		4500		
	P_{CJ}	0.379×10^{10}		0.30×10^{10}	0.3×10^{10}	
	P_p	0.625×10^{10}				
	ρ_p	924				
250 kg/m^3	D	4049		3600		
	P_{CJ}	0.2478×10^{10}		0.13×10^{10}		
	P_p	0.4538×10^{10}				
	ρ_p	552				
100 kg/m^3	D	3495				
	P_{CJ}	0.5013×10^{9}				
	P_p	0.7658×10^{9}				
	ρ_p	220				
0.75 kg/m^3	D	1622	1410*	1870*		
	P_{CJ}	0.25×10^{7}	0.284×10^{7}*	0.26×10^{7}*		
	P_p	0.484×10^{7}				
	ρ_p	8				

Ref. 1—Mader, C., "Numerical Modeling of Detonation," (University of California Press, Ltd., 1979), p. 47.

Ref. 2—Wiedermann, A., "An Evaluation of Bimodal Layer Loading Effects," IITRI Report, Feb., 1990.

Ref. 3—Stanukovitch, K.P., "Physics of Explosion" (in Russian), Nauka, 1975.

3. RESULTS

Model Validation for a One Dimensional Detonation Wave Problem

The main advantage of our particle combustion model is its description of the detonation phenomenology for a wide range of explosive particle sizes and densities. We will demonstrate this capability on a set of one dimensional test problems. For these test problems we have simulated the initiation and propagation of the detonation waves in a shock tube-like setting, where the explosive particles are distributed uniformly through the shock tube volume.

Results of these simulations are summarized in Table I, which shows detonation wave velocity, peak pressure, and peak density given as a function of the average density of the

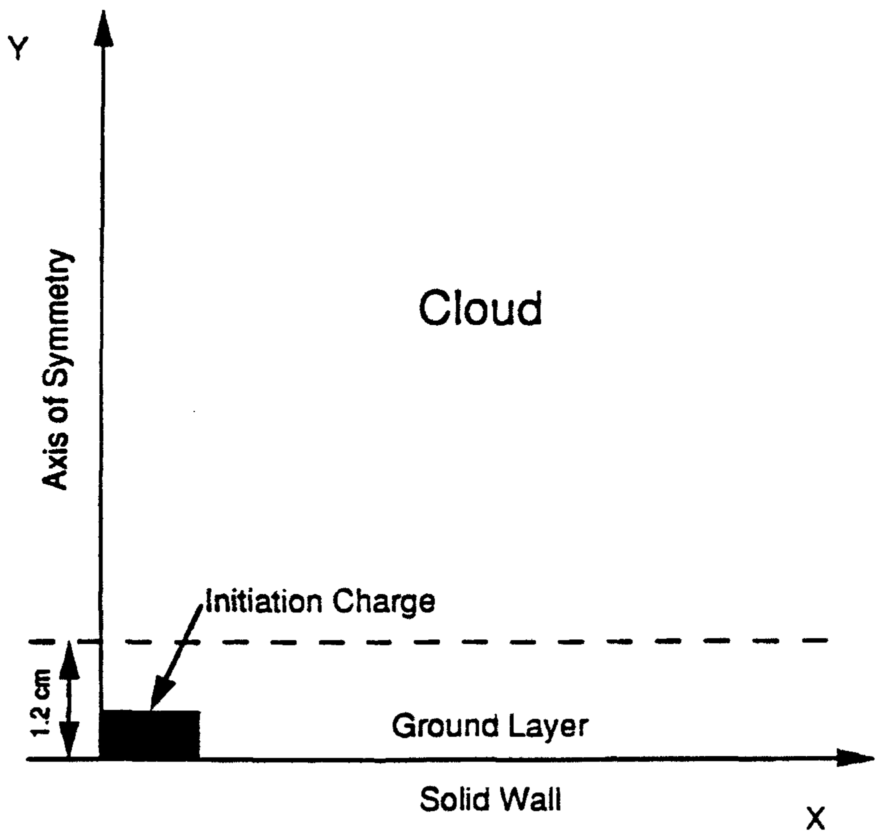


FIGURE 1 Computational domain and boundary conditions.

solid explosive. Here the explosive two-phase mixture is composed from RDX particle and air, where RDX particle concentration varies from 0.75 kg/m^3 to 1000 kg/m^3. This concentration variation covers the whole range of solid explosive concentrations of interest to our problem. The simulations performed with the MPHASE code were compared with the experimental results (Mader, 1979; Stanukovitch, 1975), and calculations were done with the TIGER code presented by Wiedremann (1990).

From Table I, it is clear that our simulation results compare favorably with other simulation results and experimental data. The maximum deviation between our results and referenced results is no greater than 15% for the entire range of explosives densities. Considering that our results were obtained with a single model for particle combustion applied to the extreme range of densities, our model gives an excellent prediction of the detonation wave parameters.

Two Dimensional Simulation Results

Figure 1 shows a setup for a typical two dimensional simulation. Here the computational domain is 25cm × 25cm. The explosive powder density is distributed according to the 4th power law of vertical distance, starting from the ground where the density is 800 kg/m^3, and rising to 1.2cm, where the density is 0.75 kg/m^3. From this point to 25cm height, the density is constant and equal to 0.75 kg/m3. The density distribution in the

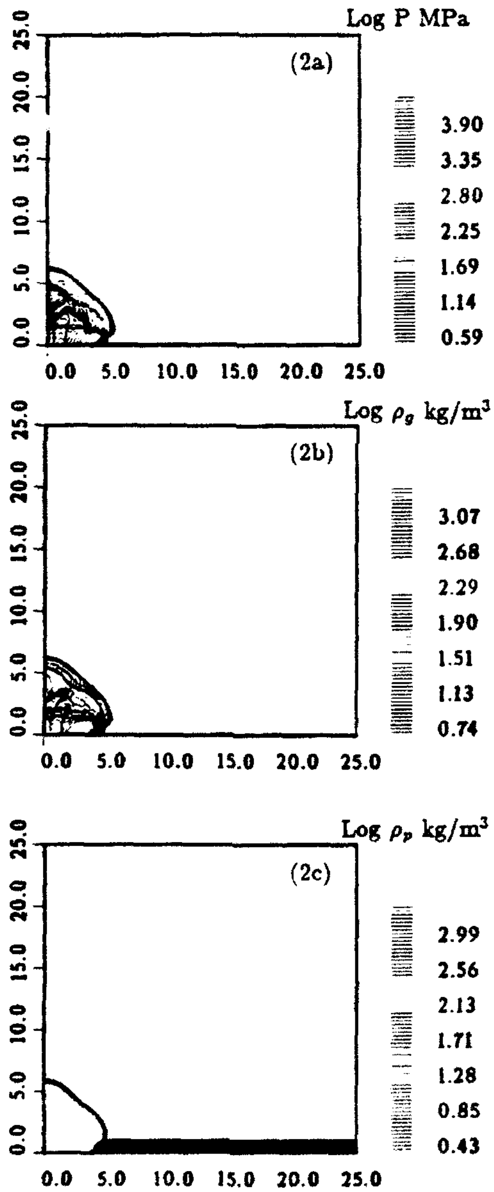


FIGURE 2 Fourth power distribution of particle density in the layer. The maximum density in the layer is 800 kg/m^3. (2a), (2b), and (2c) are gas pressure, gas density, and particle density at 12 μsec, respectively. See COLOR PLATE IV.

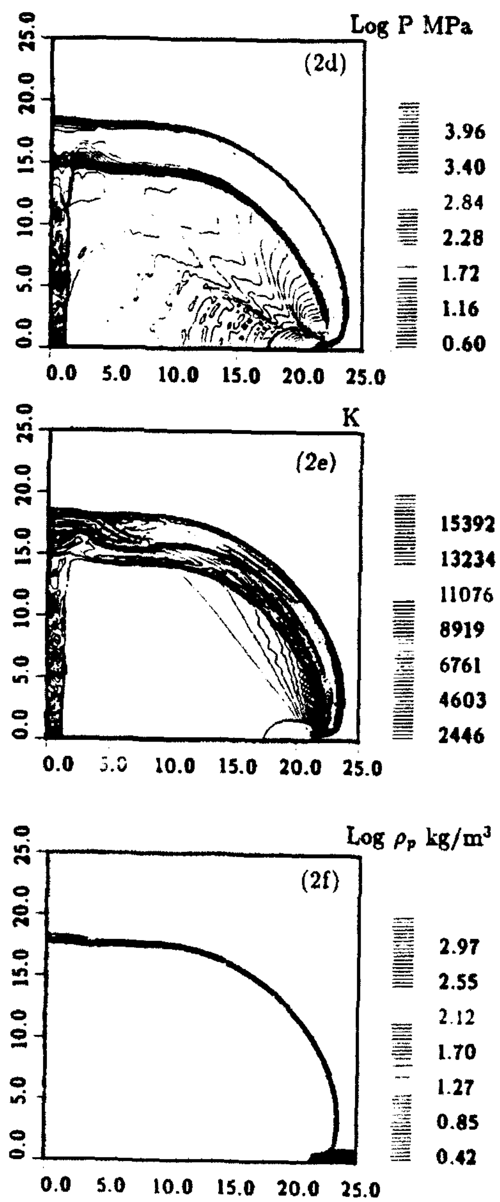


FIGURE 2 (Continued) (2d), (2e), and (2f) are gas pressure, temperature, and particle density at 55 μsec, respectively. See also COLOR PLATE IV.

direction of the "x" axis is uniform. The boundary conditions for the computational domain shown in Fig. 1 are specified as follows: solid wall along the "x" axis; symmetry conditions along the "y" axis; supersonic outflow for upper boundary and at the right of the computational domain. The mixture consists of RDX powder and air at ambient conditions and it is assumed to be quiescent at the time of initiation.

The simulation starts at t = 0 when the mixture is initiated at the lower left corner of the computational domain by an initiating charge, as shown in Fig. 1. The initiating charge is 6 mm × 10 mm, with pressure of 4 GPa and density of 450 kg/m^3. The energy released by the initiating explosion leads to formation of the detonation wave propagating through the multiphase media. Figure 2a shows pressure contours for the propagating detonation wave at the time of t = 12 μsec after initiation. Here the pressure contour levels are shown on logarithmic scale in MPa. The maximum pressure value of 7940 MPa is observed in the layer of condensed explosive located near the ground. The pressure in the layer is two to three orders of magnitude higher than pressure behind the detonation wave in the 0.75 kg/m^3 RDX cloud and air located above the distance of 1.2cm from the ground. Figure 2a demonstrates that the detonation wave in the cloud is overdriven, since the pressure behind the shock continuously rises and reaches its maximum in the layer. From this figure, we also observe that the overdriven wave propagates faster in the cloud than in the layer. This is explained by the fact that it is easier to compress air that is very lightly loaded with particles and located above the ground layer, than it is to compress air heavily loaded with a particle mixture near the ground. It is interesting to note a discontinuous pressure change between the yellow contours and the light blue and green contours behind the detonation front. This discontinuity is over-emphasized by our presentation of contour lines on the logarithmic scale; however, further examination of our simulation results indicates this feature is real and is similar in nature to barrel shocks observed for strong jets.

In Fig. 2b, gas phase density contours are shown for the time t = 12 μsec. Here the contour lines are distributed on logarithmic scale. The main features of the shock wave structure are very similar to those observed in the pressure contours figure. Here we see that a jet of high density gases reflects from the center of symmetry axis, creating a contact discontinuity that we will observe at later times. The barrel shock is clearly visible in this figure. In Fig. 2c, the particle density contour plots are shown for t = 12 μsec. The contour levels in Fig. 2c are given on the logarithmic scale and the initial deposition of the explosive material in the ground layer of the computational domain can be clearly observed. The black contour lines delineate the beginning and the end of the reaction zone in the cloud. To the left of these contours lies an area with combustion products and to the right unburned particles in the cloud. Here we can see that the reaction zone length is of the order of 1cm.

Figure 2d shows pressure contours for the same simulation for the time t = 55 μsec just before the detonation wave leaves the computational domain. In this figure we see that the global structure of the wave did change slightly from Fig. 2a. We observe that the barrel shock wave is fully developed and has a half ellipse shape. The detonation wave in the cloud is still overdriven; however, part of the shock wave front that propagates vertically becomes weaker as it gets further away from the detonation front in the layer. In Fig. 2e, gas temperature contours are shown at t = 55 μsec. In this case, it is interesting to note that the highest temperatures are observed behind the front of the overdriven cloud detonation wave in immediate vicinity of the layer's upper strata. Very high temperatures in this region can be explained by the high pressure generated from the detonation of the explosive material in the layer and by relatively low density of cloud strata in the layer's immediate vicinity. Here, as in the pressure contours graph, the area of barrel shock can be clearly identified.

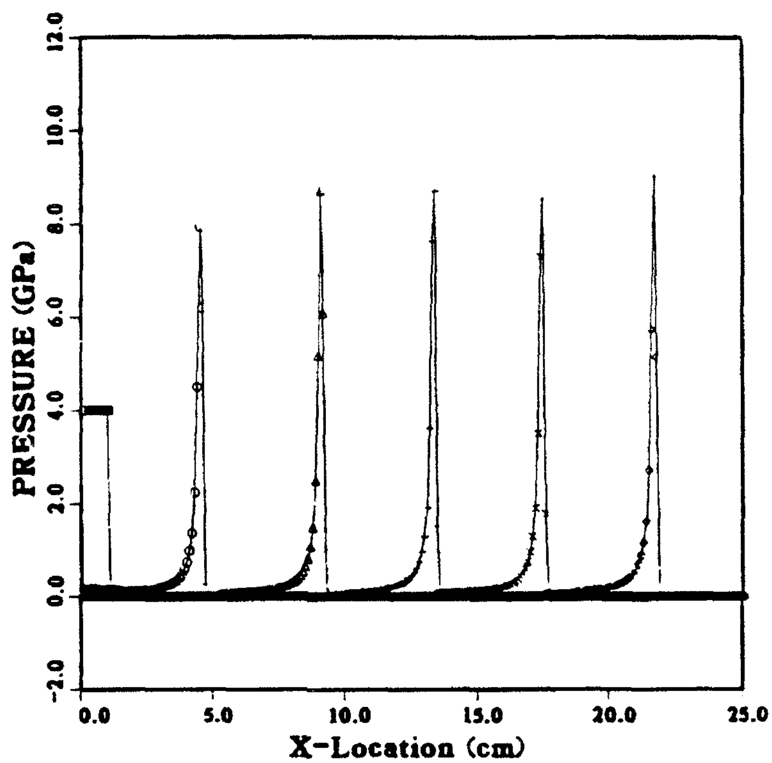


FIGURE 3 History of pressure distribution on the ground from initiation to steady detonation; □ - 0 μsec, o - 12 μsec, △ - 24 μsec, + - 34 μsec, x - 44 μsec and ◇ - 55 μsec.

We also observe in Fig. 2 a clear development of two detonation fronts, one moving vertically in the cloud and another moving horizontally in the layer. Because the energy density of the explosive particle in the layer is about three orders of magnitude larger than it is in the cloud, the vertical parts of the front represent an overdriven detonation wave in the cloud. Even though the vertical front has slowed down compared with the horizontal front, its speed and parameters far exceed those typical for detonation waves in a cloud. In fact, the self-sustained detonation regime in the cloud will develop at the distance of about three meters from the layer. The area of the front close to the detonation wave in the layer will remain hot and overdriven, since it is located very close to the detonation front in the layer. In Fig. 2f, particle density contours are shown on a logarithmic scale. We can clearly observe the reaction zone delineated by black contour lines. In this case, the reaction zone length in the cloud is about 1cm. Consistent with the gradual transition from overdriven to self-sustained detonation, the reaction zone length is larger for the vertical part of the detonation front. The detonation wave velocity observed in our simulation is approximately 4048 m/sec, which is significantly lower than the detonation wave velocity observed in RDX with a density of 860 kg/m^3 (see Table I), which is the highest density in the ground layer. This can be explained by a high gradient of particle density distribution in the layer, where the density drops rapidly from 800 kg/m^3 at the bottom of the layer to 0.75 kg/m^3 at the top strata of the layer at 12 mm above the ground.

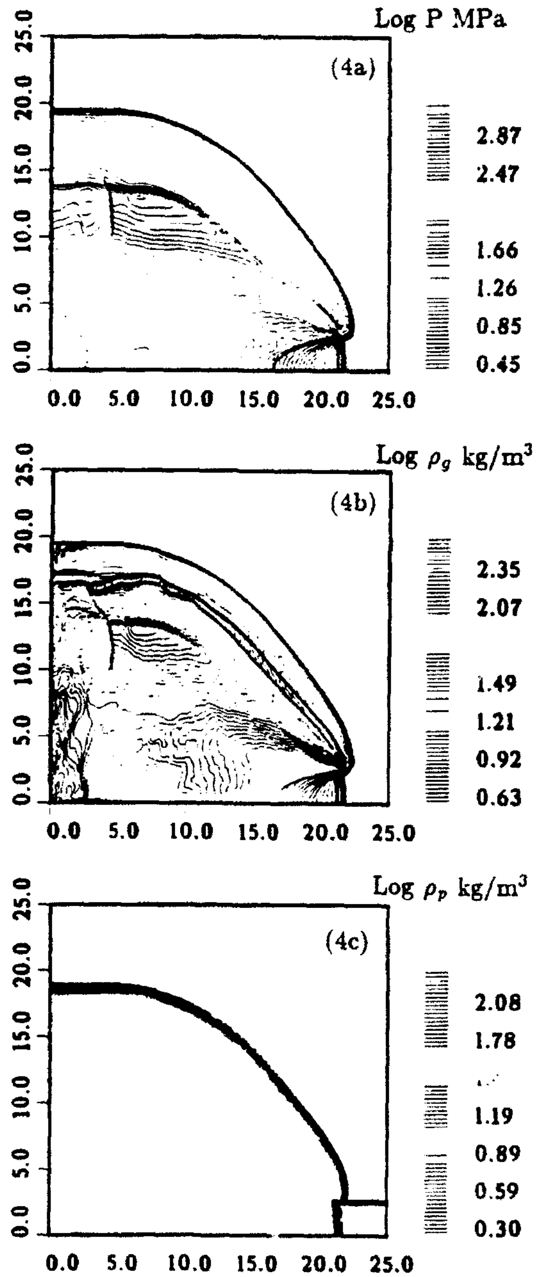


FIGURE 4 2.5 cm thick layer at constant density of 100 kg/m^3. Density in the cloud is 0.75 kg/m^3. (4a), (4b), and (4c) are gas pressure, gas density, and particle density at 66 μsec, respectively. See COLOR PLATE V.

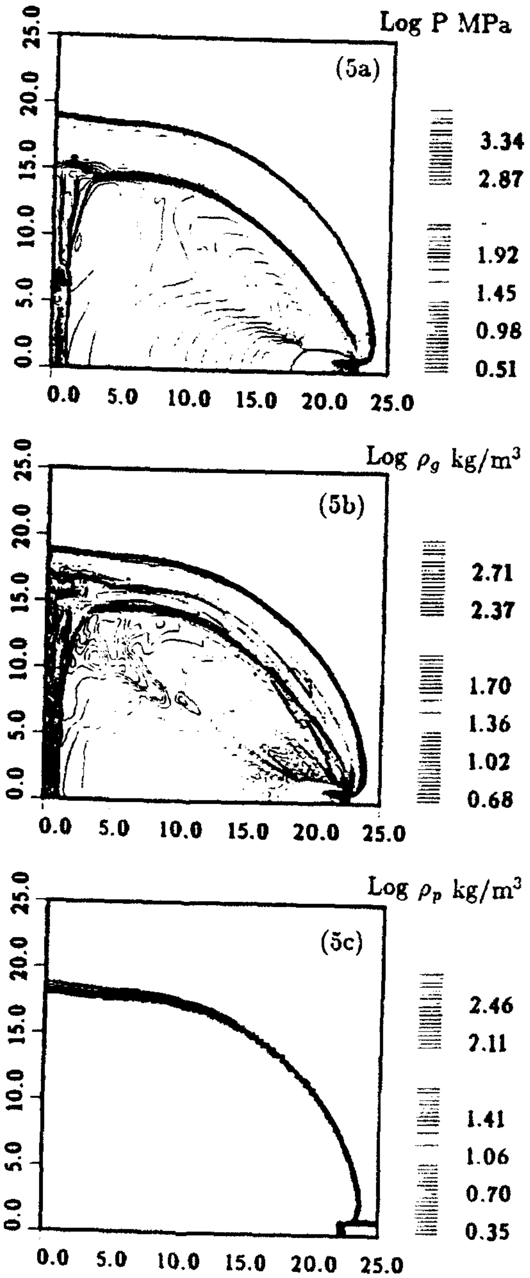


FIGURE 5 1.2 cm thick particle layer at constant density of 250 kg/m^3. Particle density in the cloud is 0.75 kg/m^3. (5a), (5b), (5c) are gas pressure, gas density, and particle density at 65 μsec, respectively. See COLOR PLATE VI.

To show the transient process from initiation to steady-state detonation, we plot the pressure-distance profiles at six separate times after ignition (Fig. 3). Here the pressure is taken on the ground. Examining the profiles, we observe that the steady detonation is reached after 10cm. For each profile, we see that the pressure distribution is characterized by a strong detonation front followed by a fast expansion wave because of lateral expansion.

To further explore properties and phenomenology of the detonation waves propagating in the layer/cloud systems, we simulated additional cases in which explosive powder density distribution was different from the case reported above, although total weight of particle per unit area remained the same.

In Fig. 4, results are shown for the case of a uniform 2.5 cm thick layer of RDX with density of 100 kg/m^3, and a 0.75 kg/m^3 cloud initiated under the same conditions as in the previous example. Figures 4a, 4b, and 4c show pressure, gas density, and particle density contour plots at t = 66 μsec. Here we observe that because the layer has much less density than the case reported above, the precursor effect of the detonation wave in the cloud preceding the wave in the layer is less pronounced. We also observe a significant difference in the shape of the strong contact discontinuity in the region of the shock front close to the layer. In Fig. 4b, we can clearly distinguish two contact surfaces, one between condensed explosive detonation products in the layer and in the cloud, and another between the detonation products from layer explosive detonation and from cloud particle detonation. We should note that these contact surfaces are over-emphasized by the logarithmic display of the contour plot levels. The maximum pressure observed in this simulation is 955 MPa, which is about one order of magnitude smaller than in the previous simulation. This is consistent with one order of magnitude difference in the maximum density of the ground layer in the two cases. The detonation wave speed (3407 m/sec) for the case presented in Fig. 4, which is only slightly lower than the speed predicted by the one dimensional simulations presented in Table I, reflects the influence of the two dimensional expansion on the detonation wave propagation.

Figure 5 presents results for the case of a uniform density of 250 kg/m^3 in a 1.2 cm ground layer. All other parameters are the same as in the previous two cases. In Figs. 5a, 5b, and 5c, pressure, gas density, and particle density contour plots are shown at the time t = 65 μsec after detonation wave initiation. Here, the detonation wave propagates faster than in the previous cases U = 3660 m/sec. This is about 400 m/sec slower than in the case of fourth power density distribution. Maximum pressure on the ground is 2150 MPa, which is consistent with the increase of powder density in the layer. The basic structure of the detonation front and the contact surfaces is similar to the case of fourth power density distribution.

4. CONCLUSIONS

We presented a mathematical model and numerical solution for the simulation of detonation wave initiation and propagation in multiphase mixtures consisting of solid combustible particles and gas. Using this model, we studied detonations in mixtures of solid RDX particles and air, with the objective of examining the effects of wide variation in particle density distribution on the dynamics and structure of detonation waves. We considered a physical system of solid particle clouds in air where a significant amount of particle can settle on the ground and the particle phase concentrations in the particle/air mixture can range from 0 to 1000 kg/m^3. This range of solid phase densities necessitated development of the model and its numerical implementation for a wide range of particle concentrations. Our validation study has shown good agreement between the simulations and referenced results for the whole range of particle concentrations.

Two dimensional simulations were done for the system of low particle density concentration clouds and ground layers formed by high concentrations of the RDX powder. We examined three cases of ground layer density distribution: a fourth power distribution within 12 mm above ground with a maximum density on the ground of 800 kg/m^3; a uniform 25 mm thick layer with a density of 100 kg/m^3; a 12 mm thick uniform layer with a density of 250 kg/m^3. In all these cases, the weight of condensed phase per unit area was the same, which allowed examination of the effects of the particle density distribution on detonation wave parameters.

In all examined two dimensional cases, the detonation wave in the cloud in the computational domain was significantly overdriven and did not play an important role. We estimated that the self-sustained regime of the detonation wave in the cloud for the examined cloud concentrations can occur only at the distances of 2–3 M above ground. At the same time, the particle density distribution in the layer determines the dynamics of the detonation wave as well as the pressure on the ground.

We observed in all three two dimensional simulations a very distinct shape of the detonation wave front in the vicinity of the layer. In this area, the overdriven detonation in the cloud is preceding the detonation wave in the ground layer. This feature of the detonation front can be explained by the fact that the energy released in the ground layer detonation wave produces a faster propagating shock wave in the dilute cloud than in the ground layer which is heavily loaded with solid particles. However, these structures were not observed experimentally, and more studies are needed to examine their parameters.

The maximum pressure affecting the ground was directly related to the maximum particle density in the lower strata of the layer. However, the detonation front velocity for the fourth power distribution case was considerably lower than calculated for a one dimensional case with 860 kg/m^3 particle density, reflecting the significant effect of two dimensional expansion. Two other cases with 250 kg/m^3 and 100 kg/m^3 maximum densities had detonation wave velocity only slightly lower than the one dimensional simulations of the same RDX/air concentrations. It is interesting to compare the simulation of the fourth power density distribution case and the 250 kg/m^3 case. In both, the same amount of explosive was distributed in the same physical space; however, the parameters of developed detonations were vastly different. Existence of the high density strata at the bottom of the ground layer in the fourth power case significantly increased the maximum pressure at the ground, and produced higher detonation wave velocity.

Using a variable density layer, we can reach a combination of pressure and velocity conditions outside of the Chapmen-Jougett limitations. The range of conditions that can be obtained in the variable density system and its parametrics needs a more systematic study. In this article, we introduced only the mathematical formulation and numerical simulation method validated for the range of conditions of interest. In addition, we have given some examples of the method's application for two dimensional simulations. However, this methodology should be linked to an experimental study for a more in-depth analysis of the phenomenology discussed here.

ACKNOWLEDGMENTS

The work reported here was partially supported by DARPA and AFOSR under contract no. F49620-89-C-0087. The authors would like to thank Col. J. Crowley and Dr. A. Nachman for their interest in this project.

REFERENCES

Burcat, A., Eidelman, S., and Manheimer-Timnat, Y. (1978). "The Evolution of a Shock Wave Generated by a Point Explosion in a Combustible Medium," *Symp. of High Dynamic Pressures (H.D.P.)*, Paris, 347.

Colella, P. (1985). "A Direct Eulerian MUSCL Scheme for Gas Dynamics," *SIAM J. Stat. Comput.*, 6, 104.

Colella, P. and Glaz, H.M. (1985). "Efficient Solution Algorithms for the Riemann Problem for Real Gases," *J. Comput. Physics*, 59, 264.

Cowan, R.D., and Fickett, W. (1956). "Calculation of the Detonation Products of Solid Explosives with the Kistiakowsky-Wilson Equation of State." *Journal of Chemical Physics*. **24**, 932.

Drake, R.M., Jr. (1961). "Discussions on G.C. Vliet and G. Leppert: Forced Convection Heat Transfer from an Isothermal Sphere to Water." *Journal of Heat Transfer*, **83**, 170.

Eidelman, S., Timnat, Y.M., and Burcat, A. (1976). "The Problem of a Strong Point Explosion in a Combustible Medium." *6th Symp. on Detonation, Coronado, CA, Office of Naval Research*, **590**.

Eidelman, S., and Burcat, A. (1980). "The Evolution of a Detonation Wave in a Cloud of Fuel Droplets; Part I, Influence of the Igniting Explosion." *AIAA Journal*, **18**, 1103.

Eidelman, S., Collela, P., and Shreeve, R.P. (1984). "Application of the Godunov Method and Its Second Order Extension to Cascade Flow Modelling." *AIAA Journal*, **22**, 10.

Eidelman, S., and Burcat, A. (1980). "The Mechanism of Detonation Wave Enhancement in a Two-Phase Combustible Medium." *18th Symposium on Combustion, The Combustion Institute, Waterloo, Ontario, Canada*.

Eidelman, S., and Burcat, A. (1981). "Numerical Solution of a Non-Steady Blast Wave Propagation in Two-Phase ('Separated Flow') Reactive Medium." *J. Comput. Physics*, **39**, 456.

Gordon, S., and McBride, B.J. (1976). "Computer Program for Calculations of Complex Chemical Equilibrium Compositions, Rocket Performance, Incident and Reflected Shocks and C-J Detonations." *NASA SP-273*, 1976 Revision.

Kauffman, C.W., Wolanski, P., Vral, E., Nicholls, J.A. and Van Dyke, R. (1979). "Shock Wave Initiated Combustion of Grain Dust," *Proc. of the Intl. Symp. on Grain Dust*, p. 164. Manhattan, KS.

Kuo, K. (1990). "Principles of Combustion," John Wiley and Sons, Inc.

Liu, J.C., Kauffman, C.W., and Sichel, M. (1990). "The Lateral Interaction of Detonating and Detonable Mixtures," (Private communication).

Mader C.L. (1979). "Numerical Modeling of Detonation," University of California Press, Ltd. London, England.

Oved, Y., Eidelman, S., and Burcat, A. (1978). "The Propagation of Blasts from Solid Explosives to Two-Phase Medium," *Propellants and Explosives*, **3**, 105.

Reinecke, W.G., and Waldman, G.D. (1975). "Shock Layer Shattering of Cloud Drops in Reentry Flight." *AIAA Paper*, 75-152.

Schlichting, H. (1983). "Bounday Layer Theory," 7th ed. McGraw-Hill.

Stanukovitch, K.P. (1975). "Physics of Explosion" (in Russian), Nauka.

Wiedermann, A. (1990). "An Evaluation of Bimodal Layer Loading Effects," *IITRI Report*, February.

AIAA 93-2940

Computation of Shock Wave Reflection and Diffraction Over a Semicircular Cylinder in a Dusty Gas

X. Yang, S. Eidelman, and I. Lottati
Science Applications International Corporation
McLean, VA 22102

AIAA 24th
Fluid Dynamics Conference
July 6-9, 1993 / Orlando, FL

COMPUTATION OF SHOCK WAVE REFLECTION AND DIFFRACTION OVER A SEMICIRCULAR CYLINDER IN A DUSTY GAS

Xiaolong Yang,* Shmuel Eidelman,† and Itzhak Lottati*
Science Applications International Corporation

Abstract

The unsteady shock wave reflection and diffraction generated by a shock wave propagating over a semicircular cylinder in a dusty gas are studied numerically. The mathematical model is a multi-phase system based on a multi-fluid Eulerian approach. A Second Order Godunov scheme is used to solve the gas phase Euler equations and an upwind scheme is used to solve the particle phase conservation equations on an unstructured adaptive mesh. For the validation of the model, the numerically predicted one dimensional shock wave attenuation is compared with experimental results. Shock wave reflection and diffraction over a semicircular cylinder in a pure gas flow is simulated first to show the excellent agreement between the present computation and the experimental results. For a shock wave reflection and diffraction in a dusty gas, the effects of particle size and particle loading on the flow field are investigated. Gas and particle density contour plots are presented. It has been shown that the shock wave configuration differs remarkably from pure gas flow depending on the particle parameters. The difference is explained as the result of momentum and heat exchange between the two phases.

Introduction

Shock wave propagation into a gas particle suspension medium has attracted great attention in recent years due to its many engineering applications. Some of these applications include blast wave propagating over a dusty surface, exhaust from a solid propellant rocket, and coal or grain dust detonation. Many studies dealing with two phase environment can be found in literature. A general description and theoretical analysis of such flow can be found in review papers by Marble[1] and by Rudinger,[2] and in a book by Soo.[3] Numerical models for dilute gas-particle flows were reviewed by Crown.[4] Numerical studies of gas-particle flow in a solid rocket nozzle can be found in Refs. 5 and 6. Miura and Glass [7] theoretically and numerically studied the oblique shock waves in a dusty-gas flow over a wedge. The one-dimensional unsteady structure of shock waves propagating through a gas-particle mixture was investigated both experimentally and numerically by Sommerfeld.[8] Recently, Kim and Chang[9] illustrated a numerical simulation of shock wave propagation into a dusty gas and the reflection of the wave from a wedge. Shock wave ignition of different reactive dust is experimentally investigated by Sichel *et al.*[10] and comprehensive model for the structure of dust detonations is also described by Fan and Sichel.[11]

In this paper, we study shock wave reflection and diffraction over a semicircular cylinder in a dusty gas. We numerically simulate the problem of a shock wave initiated in a pure gas section moving into a dusty region and impinging on a semicircular cylinder. We first formulate the compressible two-phase flow on the basis of a Eulerian multi-fluid formulation. We consider the two phases (i.e., gas and particle) to be interpenetrating continua. The dynamics of the flow are governed by conservation equations of each phase and the two phases are coupled by interactive drag force and heat transfer. We solve the system of conservation equations numerically on an unstructured adaptive grid. The objectives of the study are: (a) to solve the two-phase compressible flow field and compare the simulation with available experimental results; (b) to observe and investigate the reflection and diffraction wave patterns when a shock wave propagates over a semicircular cylinder in a dusty gas, with particle radius and loading as parameters.

The outline of this paper is as follows. Section 2 gives a description of the mathematical model and method of numerical solution, including governing conservation equations for two phases, the constitutive laws, the initial and boundary conditions, and particle parameter. A brief outline of numerical schemes and the adaptive unstructured grid is also given. In Section 3, we present our numerical simulation results. We validate our model by comparing a one-dimensional simulation of a shock wave propagating into a dusty gas with available experimental results. We also show the excellent agreement between our two-dimensional gas-only simulation with existing experimental results. Results for reflection and diffraction of shock wave over a semicircular cylinder are given for different particle parameters. Concluding remarks are given in Section 4.

---

*Drs. Yang and Lottati are Research Scientists with Science Applications International Corp. (SAIC), 1710 Goodridge Dr., MS 2-3-1, McLean, VA 22102. † Shmuel Eidelman, Research Scientist, SAIC, Associate Fellow AIAA

Mathematical Model and the Numerical Solution

Conservation Equations

The mathematical model consists of conservation governing equations and constitutive laws that provide closure for the model. The basic formulation adopted here follows the gas and dilute particle flow dynamics model presented by Soo.[3] The following assumptions are used during the derivation of governing equations:

(1) The gas is air and is assumed to be ideal gas;

(2) The particles do not undergo a phase change because for particles considered here (sand) phase transition temperature is much higher than the temperatures typical for the simulated cases;

(3) The particles are solid spheres of uniform diameter and have a constant material density;

(4) The volume occupied by the particles is negligible;

(5) The interaction between particles can be ignored;

(6) The only force acting on the particles is drag force and the only heat transfer between the two phases is convection. The weight of the solid particles and their buoyancy force are negligibly small compared to the drag force;

(7) The particles have a constant specific heat and are assumed to have a uniform temperature distribution inside each particle.

Under the above assumptions, distinct equations of continuity, momentum, and energy are written for each phase. The interaction effects between the two phases are listed as the source terms on the righthand side of the governing equation. The two-dimensional unsteady conservation equations for the two phases can be written in the vector form in Cartesian coordinates:

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = S. \qquad (1)$$

Here U is the vector of conservative variables, F and G are fluxes in x and y direction, respectively, and S is the source term for momentum and heat exchange. The definition of these vectors are:

$$U = \begin{vmatrix} \rho \\ \rho u \\ \rho v \\ e \\ \rho_p \\ \rho_p u_p \\ \rho_p v_p \\ e_p \end{vmatrix}, \quad F = \begin{vmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ u(e+p) \\ \rho_p u_p \\ \rho_p u_p^2 \\ \rho_p u_p v_p \\ u e_p \end{vmatrix}, \quad G = \begin{vmatrix} \rho v \\ \rho u v \\ \rho v^2 + p \\ v(e+p) \\ \rho_p v_p \\ \rho_p u_p v_p \\ \rho_p v_p^2 \\ v_p e_p \end{vmatrix},$$

$$S = \begin{vmatrix} 0 \\ -f_x \\ -f_y \\ -q - u_p f_x - v_p f_y \\ 0 \\ f_x \\ f_y \\ q + u_p f_x + v_p f_y \end{vmatrix}$$

where ρ, u, v, and e are gas density, velocities, and energy, respectively; ρ_p, u_p, v_p and e_p are particle density, velocities, and energy, respectively; (f_x, f_y) and q denotes drag force components acting on the particles and heat transfer to the particles, respectively. The gas pressure p is related to ρ, u, v and e for by

$$p = (\gamma - 1)[e - 0.5\rho(u^2 + v^2)] \qquad (2)$$

where γ is the specific heat ratio. The gas temperature can be found through the equation-of-state for ideal gas

$$p = \rho R T \qquad (3)$$

where R is the gas constant.

The particle temperature T_p is calculated through relation

$$e_p = \rho_p c_p T_p + 0.5\rho_p(u_p^2 + v_p^2). \qquad (4)$$

The source terms on the righthand side of Eq. (1) are momentum and heat exchange between gas and particle phases. If we let r_p and ρ_s be the particle radius and material density, respectively, then the drag forces are

$$\begin{pmatrix} f_x \\ f_y \end{pmatrix} = \frac{3}{8}\frac{\rho_p \rho}{\rho_s r_p} C_d \left[(u - u_p)^2 + (v - v_p)^2\right]^{1/2} \begin{bmatrix} (u - u_p) \\ (v - v_p) \end{bmatrix}. \qquad (5)$$

The particle drag coefficient C_d depends on relative Reynolds number, Re and relative Mach number, M_r. In the present study, since the relative Mach number is small ($M_r < 0.5$), the effect of M_r on C_d is neglected. The Reynolds number, Re, is based on the relative velocity between the gas and particle phases. After testing the drag coefficients given by Sommerfeld[8] and by Clift *et al.*,[12] the following two were adopted:

$$C_d = \frac{24}{Re}(1 + 0.15 Re^{0.687}) \text{ for } Re < 800.$$

and

$$C_d = \frac{24}{Re}(1 + 0.15 Re^{0.687}) + \frac{0.42}{1 + 42500 Re^{-1.16}} \quad \text{for } Re > 800 . \qquad (6)$$

Here the Reynolds number Re is defined as

$$Re = \frac{2\rho r_p[(u-u_p)^2 + (v-v_p)^2]^{1/2}}{\mu} \quad (7)$$

Viscosity, μ, is calculated at film temperature, namely, $T_f = 0.5(T_p + T)$, and the temperature dependency of the viscosity is evaluated according to Sutherland's law

$$\mu = \mu_r \left(\frac{T}{T_r}\right)^{3/2} \frac{T_r + \Phi}{T + \Phi} \quad (8)$$

where μ_r is the dynamic viscosity of the gaseous phase at the reference temperature and Φ is an effective temperature, called the Sutherland constant.

The rate of heat transfer from gaseous phase to the particle phase is given by

$$Q = \frac{3}{2} \frac{\rho_p}{\rho_s} \frac{\mu C_v}{Pr} Nu\,(T - T_p) \quad (9)$$

where $Pr = \mu c_p/k_g$ is the Prandtl number, and c_p and k_g are the specific heat and thermal conductivity of gas, respectively. The Nusselt number Nu is a function of Reynolds number and the Prandtl number as given by Drake[13]

$$Nu = \frac{2r_p h}{R} = 2 + 0.459 Re^{0.55} Pr^{0.33}. \quad (10)$$

Initial and Boundary Conditions

The geometry of the computational domain is shown in Fig. 1. The initial conditions for gas are $\rho_o = 1.2kg/m^3$ and $p_o = 101.3kpa$, with a coming shock at x = -0.5. There are no particles from $-1.0 \le x \le 0.0$. From $x \ge 0.0$, particles are initially in thermal and kinematic equilibrium with surrounding gas. The particles that are uniformly distributed in the dusty region have the following parameters for different test problems:

Mass loading, ρ_p: 0.25 kg/m^3, 0.76 kg/m^3;
Mass material density, ρ_s: 2500 kg/m^3;
Particle radii, r_p: 10 μm, 25 μm, 50 μm;
Specific heat, c_s: 766 $J/kg/K$.

The lower boundary and cylinder surface are solid walls and assumed adiabatic and impermeable. A reflecting boundary condition is assumed for both the gas and particle phase. Particles are assumed to experience a perfect elastic collision with the wall and reflect from the wall. The right and upper boundaries are open boundaries where a nonreflection boundary condition is used for the gas phase and a zero normal gradient condition is used for particle phase.

Numerical Method of Solutions

The system of partial differential equations described in the previous paragraph is integrated numerically. Equation (1) is repeated here:

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = S. \quad (1)$$

In order to solve this equation numerically, an operator time-splitting technique is used. Assuming that all flow variables are known at a given time, we can calculate its advancement in time by splitting the integration into two stages.

In the first stage, the conservative part of Eq. (1) is solved:

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = 0. \quad (11)$$

The Second Order Godunov method is used for the integration of the subsystem of equations describing the flow of the gaseous phase (first four components of Eq. (1)). The method is well documented in literature.[14,15,16] The subsystem of equations describing the particle phase flow is integrated using a simple first order finite difference upwind scheme[17]. This is done because there is no shock in the particle phase and the upwind scheme leads to a robust and accurate integration scheme.

In the second stage, the source term is added and the following equation is solved:

$$\frac{\partial U}{\partial t} = S. \quad (12)$$

To integrate this equation in time, we need to obtain S as a function of U. We calculate S through Eqs. (5) to (10).

In order to produce a solution of the high spatial accuracy at minimal computational cost, an unstructured triangular grid with adaptive procedure is used. The adaptive procedure will automatically enrich the mesh by adding points in the high gradient (or high flow activity) region of the flow field and by removing points (coarsening mesh) where they are not needed. The dynamic nature of mesh enrichment is shown in Fig. 4 for three different time frames. One can see that a very fine mesh is generated around shock fronts and other steep density gradient regions.

Results

Model Validation for One-Dimensional Shock Wave Propagation in Dusty Gas

To test the momentum and heat exchange mechanism for the current two-phase model, we first simulate a one-dimensional problem of a normal shock wave propagating into a dusty gas. We numerically simulate the experiments conducted by Sommerfeld.[8] In the experiments, small glass spherical particles of material density ρ_s = $2500kg/m^3$, specific heat capacity c_s = $766J/kg/K$, and average diameter of 27 μm

were used as the suspension particle phase. The incoming shock Mach number M_s and particle loading ratio $\eta = \rho_p/\rho$, are two varying parameters. The experimental results and our numerical simulation results of shock Mach number as a function of distance for two test cases are shown in Fig. 2a ($\eta = 0.63, M_s = 1.49$) and Fig. 2b ($\eta = 1.4, M_s = 1.7$) for comparison purposes. It is clear that the agreement between the prediction of shock wave attenuations from our present model and the experimental results is very good.

Two-Dimensional Simulation Results for Pure Gas Flow

To test the accuracy of the two-dimensional computation, we first compute the pure gas flow case of a shock wave reflection and diffraction over a semicircular cylinder. We then compare the simulation with experimental results. Shock wave reflection on a wedge has been extensively studied by many researchers (see e.g., review papers of Ben-Dor and Dewey[18] and Hornung[19]). Shock wave reflection over circular cylinders was numerically simulated by Yang *et al.*[20]. Recently, Glass *et al.*[21] using high order Godunove scheme numerically simulated the shock wave reflection over a half diamind and semicircular cylinder and compared the simulation with experimental results obtained by Kaca.[22]. Figure 3 is a schematic sketch to show four stages of a shock wave reflection over a semicircular cylinder and terminologies which will be used to describe the flow fields. Figures 4a, 4b and 4c show the calculated density contours at three moments in time. When the planar shock wave propagates and encounters the cylinder, it first experiences a head-on collision with the front stagnation point of the semicylinder and then immediately reflects from the first quarter of the cylinder, forming a regular reflection (RR), which is shown in Fig. 4a. The regular reflection consists of two shocks, i.e., the incident shock and reflected shock, both originating from a common point on the cylinder wall. As the shock wave propagates up the cylinder, the angle between the incident shock and the tangent of the cylinder becomes larger and the regular reflection changes into a Mach Reflection (MR) as shown in Fig. 4b. The MR is characterized by three waves, incident shock (I), reflected shock (R), and Mach stem (M). All three shocks intersect at one common point called triple point (T). For Mach reflection, one can further observe both Simple Mach Reflection (SMR) and Complex Mach Reflection (CMR). Later, as the incident shock wave passes over the top of the semicircular cylinder, it experiences a rarefaction on the back side of the cylinder. The shock wave system grows upward and rightward with a curved Mach stem and forms a slipline(S) or a contact discontinuity (CD) as shown in Fig. 4c. In Figs. 5a and 5b, the interferogram from the experiment[22] and densit contours from the present simulation are compared for e same flow condition and same time. Note that the d sity levels are normalized by the ambient gas density Fig. 5. As one can see from Fig. 5, the results sh an e cellent quantitative as well as qualitative agree nt between the numerical simulation and experimen results.

Two-Dimensional Simul tion Results of Two-Phase Flow

The basic setup for the two-phase simulation is shown in Fig. 1. Here the planar shock with $Ms = 2.8$ propagates into an area of a dusty gas and impinges on a semicircular cylinder. The interface between pure air and dusty air is located at $x = 0.0$ of the computational domain. The area of the dusty air contains a semicylinder with a radius of $1m$. The size of the computational domain, initial parameters of the gas, parameters of the incoming shock, size of the semicylinder and its location in the computational domain, are the same as in the reflection and diffraction simulation presented in the previous section.

The main objective of this set of simulations is to study the effects of particle size and particle loading on the parameters of the reflected and diffracted shock waves. It is also valuable to study the dynamics of particle media, since it is extremely difficult to observe these interactions experimentally in an optically thick dusty gas.

The first set of simulation results is shown for the case with dust parameters $r_p = 10\mu m$ and $\rho_p = 0.25$ kg/m^3. The gas parameters and the parameters of the incoming shock wave are the same as in the pure gas case presented above. In Figs. 6a and 6b, particle density contours and gas density contours are shown at the stage when the incident shock wave has reached the top of the semicylinder. At this stage, the largest difference of velocity and temperature between the two phases exists and the nonequilibrium between the two phases causes extensive heat and momentum exchange between particles and the gas. The presence of the particles causes a widening of the shock that is more noticeable for the incident shock. Also, an additional contour line is observed at the dusty gas/pure gas interface. Comparing gas density for pure gas flow field shown in Fig. 4b and the dusty gas density of Fig. 6b, we see that Mach stem and contact discontinuity resulting from Mach reflection are smeared in the dusty gas flow due to the presence of the particle. The particle density contours depict significant piling up of the dust particles at the leading edge stagnation point of the cylinder.

In Figs. 6c and 6d, the particle density and gas density contours are shown at the stage where significant diffraction has taken place and the shock front is

approaching the trailing edge of the cylinder. Further widening of the shock and some smearing of the slip line that originates at the triple point is evident. The particle density contours reveal that the particles were swept by the gas flow to the area of triple point and slip line for the gas flow, leaving a small amount of particles at the leading edge. We should note that this behavior is specific for our problem, where at $t = 0$, the dusty gas area was located at $x = 0$ and there is no influx of the dust from the left boundary. Also in Fig. 6c, we note that the particles reach a distinct local maxima at the distance about 25 cm behind the incident shock front. At this maxima the particle density is 0.86 kg/m^3, which is more than three times the initial particle density. The particle density reaches a maximum value at the location of the gas slip line. We observe a significant accumulation of the particles that have been moved along the slip line by the shear flow. The larger concentration of particles in the vicinity of triple point is, in fact, the remainder of the particles that were swept up with the flow. It is also interesting to observe that an essentially particle-free zone is formed due to the effects of particles slipping over the top of the cylinder and the rarefaction wave behind the cylinder.

To study the influence of particle loading on the dynamics of reflection and diffraction, we have simulated the case with a dust density of $\rho_p = 0.76$, and with $r_p = 10\mu m$. The results for this simulation are shown in Figs. 7a and 7b in the form of particle and gas density contour plots. In Fig. 7a, the particle density contours are shown at the diffraction phase. Here we can observe two local maxima for particles accumulated in the regions along the slip line characteristic for the shock diffraction process. It should be noted here that in our problem the conditions behind the incident shock wave and its structure are in constant flux. At higher loading, dust will have a profound effect on the gasdynamics of reflection and diffraction. Figure 7b shows gas density contours for the reflection stage corresponding to the particle density contours shown in Fig. 7a. We observe from Fig. 7b that the incident shock wave is significantly smeared and the triple point cannot be clearly identified. Because of the widening of the incident shock, the area where the reflected and incident shock join is spread over 50 cm distance. From Fig. 7a, we see that the high density particle region is spread wider than in the previous case, and the particle density reaches its maximum at about 25 cm behind the front. There is a visible maximum in gas density in the area where the reflected shock is interacting with the area of maximum particle density behind the incident shock. A part of the reflected shock front that is moving to the left side of the computational domain is not affected by the dust since it is propagating into an area with little dust concentration. The parameters and structure of this part of the front remain basically the same as in the case of pure gas flow.

To examine the effect of particle size on the reflection-diffraction process, we simulated a case where the particle loading and gas flow conditions are the same as in the previous case with particle density $\rho_p = 0.76$. However, the particle size is $r_p = 50\mu m$. In Figs. 8a and 8b, results for this simulation are illustrated by particle density and gas density contours correspondingly. The particle contour plots depict a significantly wider particle relaxation zone than in the previous case. The longer relaxation zone is caused by the larger inertia of larger particles. The maximum particle density of 2.64 kg/m^3 is reached 50 cm behind the incident shock front. This value is significantly lower than 4.01 kg/m^3 reached behind the shock in calculation with 10 μm particles. Larger particles skip above the apex of the cylinder creating a void where particle density is very small. Also, because of larger particle size, the maxima of particle concentration that has been created by a slip surface of the reflected Mach stem is indistinct. The main reason for this is that the particles do not follow the gas flow as closely as they did in the previous case due to the inertia of large particles. The maximum particle density is reached here at the slip line behind the Mach stem.

Comparing gas density of Fig. 8b to the previous case shown in Fig. 7b, we observe that the slip line behind the curved Mach stem becomes less distinguishable in Fig. 7b. This result is expected, since at fixed particle loading, smaller particles have a larger surface/volume ratio and the larger surface/volume ratio increases momentum and heat exchange between the two phases.

One general comment regarding all three cases presented above: Due to the heat and momentum exchange between the two phases, the shock is decaying as it traverses the cylinder. Ultimately, it will reach a new equilibrium state as suggested by Fig. 2. It should be noted that the shock considered in the previous three cases is still in the process of transition in the gas-particle mixture.

Conclusion

In this paper, numerical study for a two-phase compressible flow is performed for the reflection and diffraction of a shock wave propagating over a semicircular cylinder in a dusty gas. The following conclusions can be made:

(1) The validation study for a one-dimensional shock wave propagating in a dusty gas shows a good agreement between the prediction of our model and the results of the experiment;

(2) For a two-dimensional gas-only flow, numerical results agree well with existing experimental data quali-

tatively and quantitatively, indicating that the gas phase is accurately simulated by the adaptive grid technique;

(3) Particles in the gas can have a profound effect on the shock wave reflection and diffraction pattern, which is a function of particle size and loading. The lesser the particle loading, the less the influence of particle on the flow field:

(4) In the three simulation cases, there is a particle accumulation behind the "back shoulder" of the semi-circular cylinder due to the effect of particle inertia and gas rarefaction wave;

(5) For different particle size at fixed particle loading, the larger particle will have a longer relaxation zone and less accumulation at "back shoulder" and behind incident shock. The gas density contours show a less distinguishable slip line in small particle case than in the large particle case.

Acknowledgments

The work reported here was partially supported by DARPA and AFOSR under contract no. F49620-89-C-0087. The authors would like to thank LtCol. J. Crowley and Dr. A. Nachman for their interest in this project.

References

[1]Marble, F., "Dynamics of Dusty Gases," *Annual Review of Fluid Mechanics*, Vol. 2, 1970, pp. 369-377.

[2]Rudinger, G., "Some Properties of Shock Relaxation in Gas Flows Carrying Small Particles," *Physics of Fluids*, Vol. 7, 1964, pp. 658-663.

[3]Soo, S. L., *Particulates and Continuum*, Hemisphere Publishing Corporation, New York, 1989, pp.266-324.

[4]Crowe, C. T., "Review-Numerical Models for Dilute Gas Particle Flow," *Journal of Fluids Engineering*, Vol. 104, Sept. 1982, pp. 297-303.

[5]Hwang, C. J. and Chang, G. C., "Numerical Study of Gas-Particle Flow in a Solid Rocket Nozzle," *AIAA Journal*, Vol. 26, No. 6, 1988, pp. 682-689.

[6]Chang, I-Shih, "Three-Dimensional, Two-Phase, Transonic, Canted Nozzle Flows," *AIAA Journal*, Vol. 28, No. 5, 1989, pp. 790-797.

[7]Miura, H. and Glass I. I., "Oblique Shock Wave in a Dusty-Gas Flow Over a Wedge," *Proceedings of Royal Society of London* A, Vol 408, 1986, pp. 61-68.

[8]Sommerfeld, M., "The Unsteadiness of Shock Waves Propagating through Gas-Particle Mixtures," *Experiments in Fluids*, Vol. 3, No. 2, 1985, pp. 197-206.

[9] Kim, S-W. and Chang, K-S. 'Reflection of Shock Wave from a Compression Corner i . Particle-laden Gas Region," *Shock Waves*, Vol. 1, No . 1991, pp. 65-73.

[10]Sichel, M., Baek, S. W., Kau nan, C. W., Maker, B. and Nicholls. "The Shock w gnition of Dusts," *AIAA Journal*, Vol. 23, No. 9, . pp. 1374-1380.

[11]Fan, B. and Sichel, M., " omprehensive Model for the Structure and Dust Dete tions," *Twenty-Second Symposium (International) on Combustion*, The Combustion Institute, PA, 1988, pp. 1741-1750.

[12]Clift, R., Grace, J. R. and Weber, M. E., *Bubbles, Drops and Particles*, Academic, New York, 1978.

[13]Drake, R.M., Jr., "Discussions on G.C. Vliet and G. Leppert: Forced Convection Heat Transfer from an Isothermal Sphere to Water," *Journal of Heat Transfer*, Vol. 83, No. 2, 1961, pp. 170-179.

[14]Eidelman, S., Colella, P., and Shreeve, R.P., "Application of the Godunov Method and Its Second Order Extension to Cascade Flow Modelling," *AIAA Journal*, Vol. 22, No. 11, 1984, pp. 1609-1615.

[15]Colella, P., "A Direct Eulerian MUSCL Scheme for Gas Dynamics," *SIAM Journal Scientifical and Statistical Computation*, Vol. 6, 1985, pp. 104-117.

[16]Colella, P. and Glaz, H.M., "Efficient Solution Algorithms for the Riemann Problem for Real Gases," *Journal of Compututational Physics*, Vol. 59, No. 3, 1985, p. 264-289.

[17]Peyret, R. and Taylor, T. D., *Computational Methods for Fluid Flow* Springer-Verlag, New-York, 1983, pp. 18-31.

[18]Ben-Dor, G. and Dewey J. M., "The Mach Reflection Phenomenon: A Suggestion for and International Nomenclature," *AIAA Journal*, Vol. 23, N. 10, 1985, pp. 1650-1652.

[19]Hornung, H., "Regular and Mach Reflection of Shock Waves," *Annual Review of Fluid Mechanics*, Vol. 18, 1986, pp. 33-58.

[20]Yang, J. Y., Liu, Y. and Lomax H., "Computation of Shock Wave Reflection by Circular Cylinder," *AIAA Journal*, Vol. 25, 1987, No. 5, pp. 683-689.

[21]Glass, I. I., Kaca, J. Zhang, D. L. Glas, H. M., Bell, J. B. and Tangenstein, J., *Current Topics in Shock Waves, 17th Int'l Symp. on Shock Tubes and Waves*, edited by Y. W. Kim, AIP Conference Proceedings 208, American Institute of Physics, New York, 1990, pp. 246-251.

[22]Kaca, J., "An Interferometric Investigation of Diffraction of a Planar Shock Wave over a Semicircular Cylinder," *UTIAS Technical Note 269*, Institute for Aerospace Studies, University of Toronto,1988.

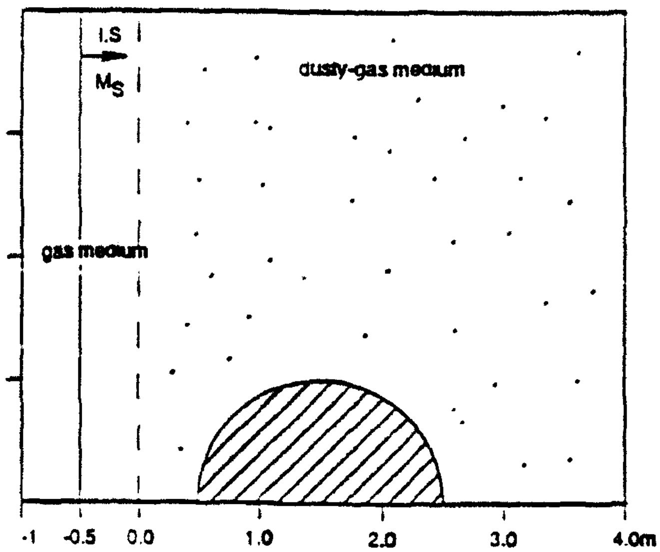


Figure 1. An illustration of the considered flow field.

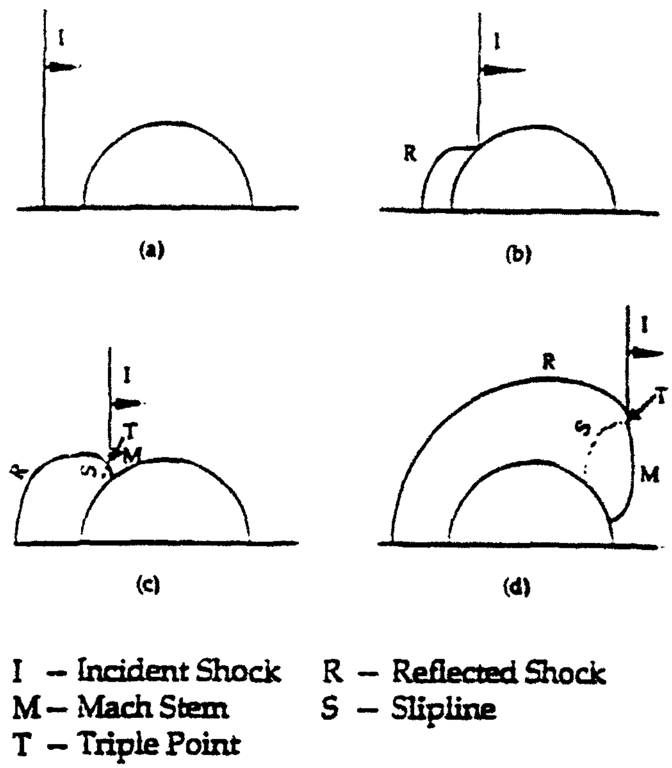


Figure 3. Stages of shock wave reflection over a semicircular cylinder, (a) before collision, (b) regular reflection, (c) Mach reflection, (d) well developed Mach reflection.

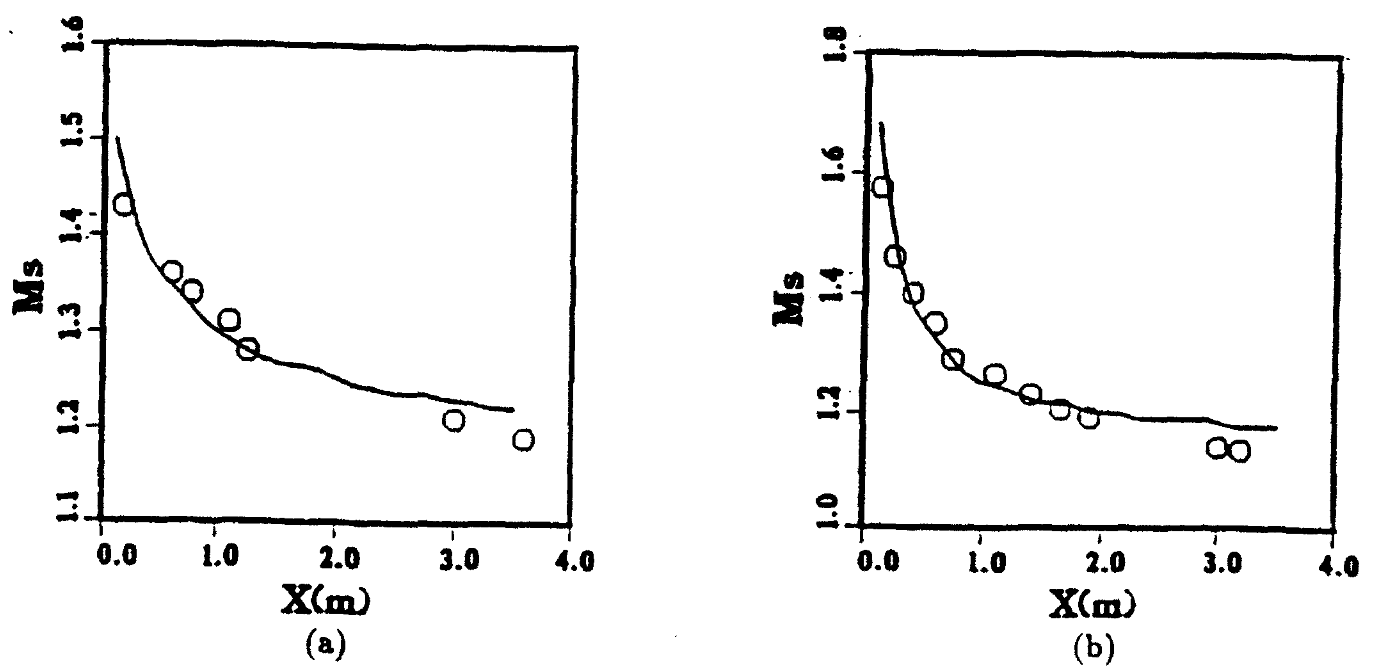
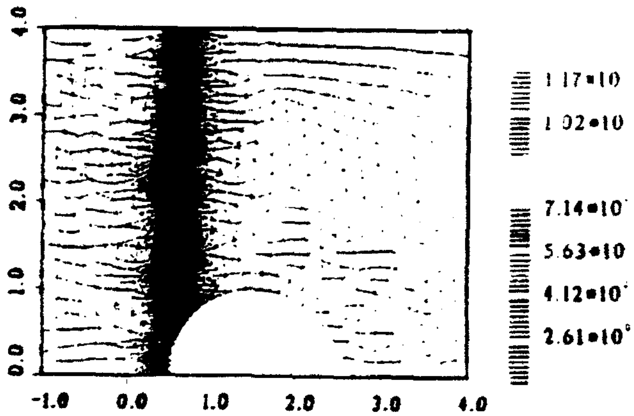
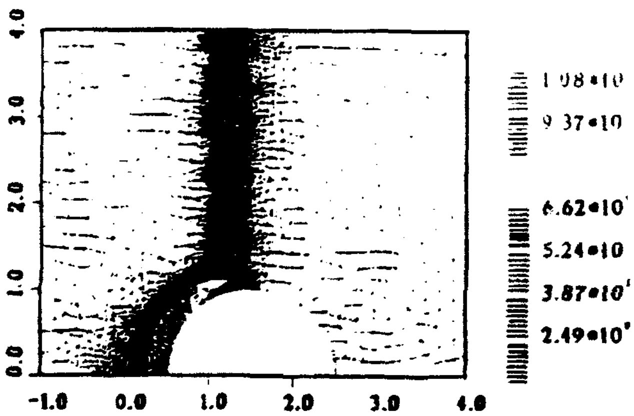


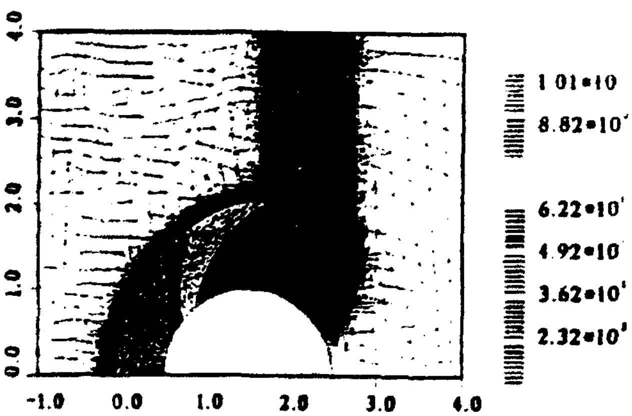
Figure 2. Comparison between computational prediction and experimental measurement of shock wave attenuation for (a) $M_s = 1.49$, $\eta = \frac{\rho_2}{\rho_o} = 0.63$ and (b) $M_s = 1.7$, $\eta = \frac{\rho_2}{\rho_o} = 1.4$ (o experiment, - calculation).



(a)

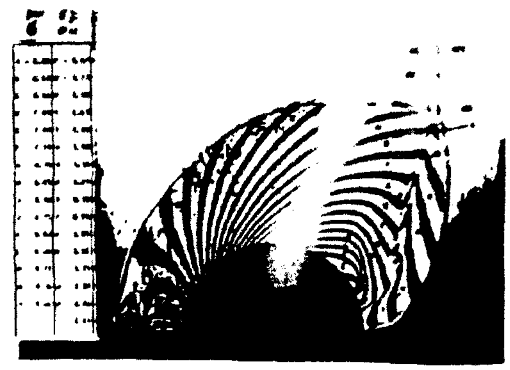


(b)

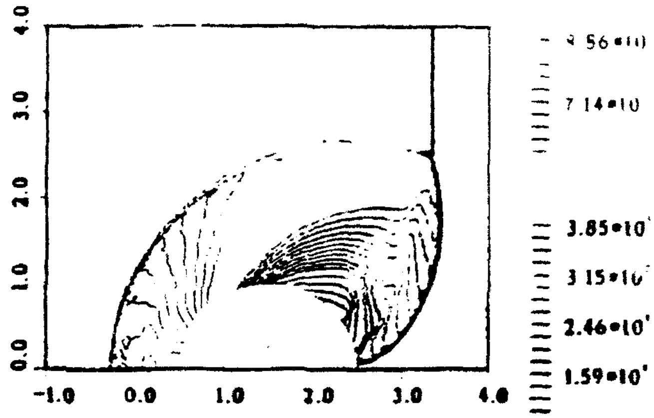


(c)

Figure 4. Computed density contours with adapted grid at three different times: (a) regular reflection (RR), (b) Mach reflection (MR) and (c) diffraction with slipline (S).

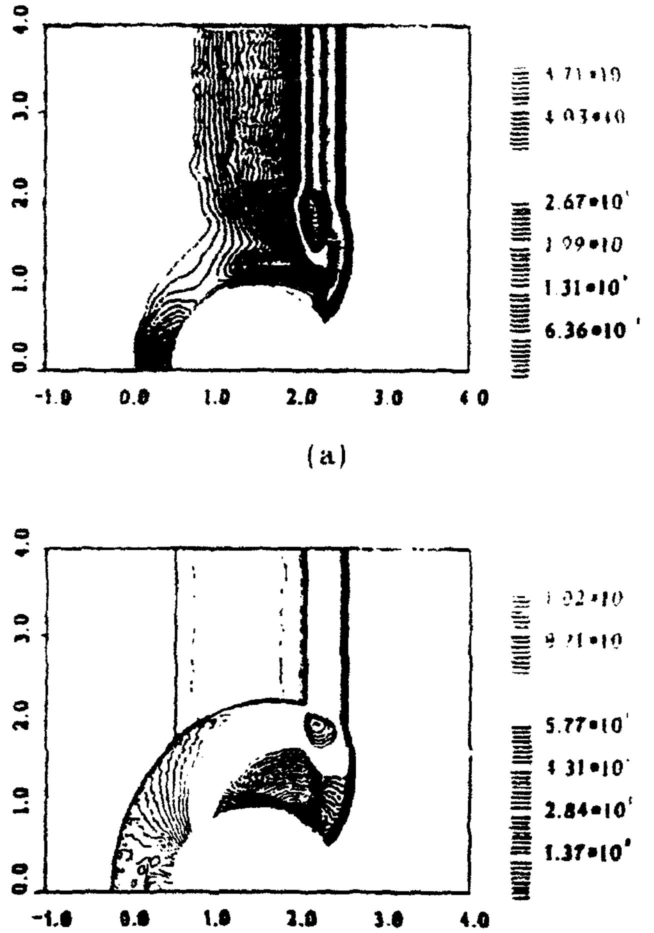


(a)



(b)

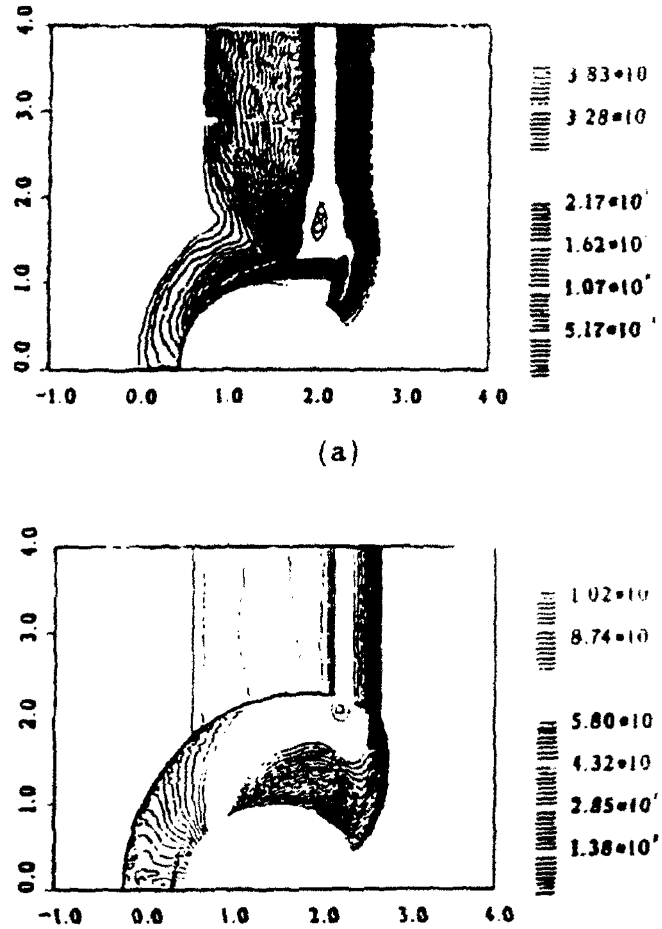
Figure 5. Comparison for M_s = 2.80 gas - only flow. (a) interferogram from experiment conducted by Kaca (1988), (b) density contours from present calculation.



(a)

(b)

Figure 7. Density contours for the case: $M_s = 2.8$, $\rho_p = 0.76$ kg/m^3 and $r_p = 10\mu$m, (a) particle density and (b) gas density.



(a)

(b)

Figure 8. Density contours for the case: $M_s = 2.8$, $\rho_p = 0.76$ and $r_p = 50\mu$m, (a) particle density and (b) gas density.

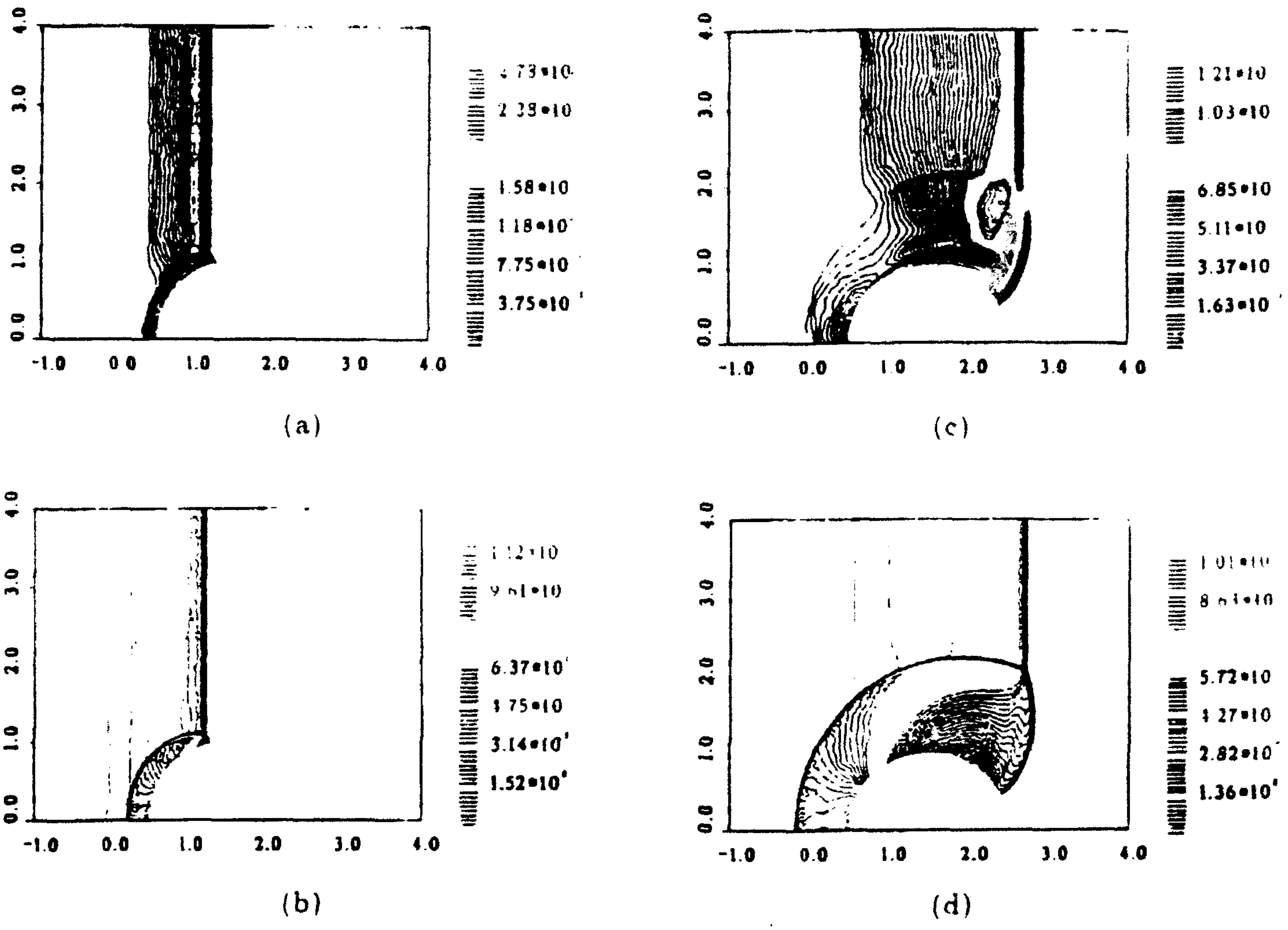


Figure 6. Density contours for the case: $M_s = 2.8$, $\rho_p = 0.25$ kg/m^3, $r_p = 10\mu$m at two different times, (a) particle density at t_1, (b) gas density at t_1, (c) particle density at t_2, and (d) gas density at t_2.

AIAA 93-3089

Acoustic Wave Focusing in an Ellipsoidal Reflector for Extracorporeal Shock-wave Lithotripsy

Itzhak Lottati and Shmuel Eidelman
Science Applications International Corporation
McLean, VA 22102

AIAA 24th
Fluid Dynamics Conference
July 6-9, 1993 / Orlando, FL

ACOUSTIC WAVE FOCUSING IN AN ELLIPSOIDAL REFLECTOR FOR EXTRACORPOREAL SHOCK-WAVE LITHOTRIPSY

by

Itzhak Lottati* and Shmuel Eidelman†
Science Applications International Corporation

Abstract

Simulations of acoustic wave focusing in an ellipsoidal reflector for extracorporeal shock-wave lithotripsy (ESWL) are presented. The simulations are done on a structured/unstructured grid with a modified Tait equation of state for water. The Euler equations are solved by applying a second-order Godunov method. The computed results compare very well with the experimental results.

Introduction

Research relating to focusing of shock and acoustic waves is of practical interest for extracorporeal shock-wave lithotripsy (ESWL). A considerable body of work is dedicated to this subject (see e.g., review in Ref. 1), and numerical simulations play a prominent role in research on these devices. It is conceivable that real-time numerical simulation can be used for better assessment of shock-wave impact on the targeted areas and more effective focusing. Requirements for these real-time simulations in terms of robustness, accuracy and efficiency are very stringent, and can be satisfied only with the most advanced numerical methods.

Structured rectangular grids allow the construction of numerical algorithms that integrate the fluid conservation equations efficiently and accurately. The efficiency of these schemes results from the extremely low storage overhead needed for domain decomposition and the efficient and compact indexing, which also defines domain connectivity. These two factors allow code construction based on a structured domain decomposition that can be highly vectorized and parallelized. Integration in physical space on orthogonal and uniform grids produces numerical algorithms with the highest possible accuracy. The disadvantage of structured rectangular grids is that they cannot be used to decompose computational domains with complex geometries. Thus it is difficult to represent computationally a complex computational domain with the curved boundaries characteristic of typical reflectors used in ESWL devices.

The early developers of computational methods realized that, for many important applications of Computational Fluid Dynamics (CFD), it is unacceptable to describe curved computational domain boundaries using the stair-step approximation available with the rectangular domain decomposition technique. To overcome this difficulty, the techniques of boundary-fitted coordinates were developed. With these techniques, the computational domain is decomposed on quadrilaterals that can be fitted to the curved domain. The solution is then obtained in physical space using the geometrical information defining the quadrilaterals, or in the computational coordinate system that is obtained by transformation of the original domain into a rectangular domain. The advantage of this technique is that it employs the same indexing method as the rectangular structured domain decomposition methods that also serve to define domain connectivity. The boundary fitted coordinate approach leads to efficient codes, with approximately a 4:1 penalty in terms of memory requirement per cell as compared with rectangular domain decomposition. However, this approach is somewhat restricted in its domain decomposition capability, since distortion or large size variations of the quadrilaterals in one region of the domain lead to unwanted distortions or increased resolution in other parts of the domain. An example of this is the case of structured body-fitted coordinates used to simulate flows over a profile with sharp trailing edges. In this case, increasing the resolution in the vicinity of the trailing edge increases resolution in the whole row of elements connected to the trailing edge elements.

The most effective methods of domain decomposition developed to overcome this disadvantage are those using unstructured triangular grids. These methods were developed to cope with very complex computational domains. The unstructured grid method, while efficient and powerful in domain decomposition, results in codes that must store large quantities of information defining the grid geometry and connectivity, and have large computational and storage overheads. As a rule, a code with an unstructured grid requires greater storage by a factor of 10, and will run about 5 times slower on a per cell per iteration basis than a structured rectangular code.

Unstructured triangular meshes are designed to pro-

*Dr. Lottati is a Research Scientist with Science Applications International Corp. (SAIC), 1710 Goodridge Dr., MS 2-3-1, McLean, VA 22102, †Dr. Eidelman, Research Scientist, SAIC, Associate Fellow AIAA

vide a grid that is fitted to the boundary of complex geometry. The flexibility of the unstructured mesh that allows complex geometry to be gridded should be weighed against the huge memory requirement needed to define the interconnectivity of the triangles. To cut down on the memory overhead, unstructured grid methods are used to their best advantage when combined with grid adaptivity. This feature usually allows the dynamic reallocation of triangles according to the physics and geometry of the problem solved, which leads to a substantial reduction in the number of cells needed for the domain decomposition. However, this advantage is highly dependent on the problem solved. Adaptive unstructured grids have an advantage over nonadaptive unstructured domain decomposition if the area of high resolution needed is around one-tenth of the global area of the computational domain. As a result, while the adaptive unstructured method may be extremely effective for simulating flow with multiple shock waves in complex geometries, it becomes extremely inefficient when high resolution is needed in a substantial area of the computational domain.

Our approach to domain decomposition for ESWL applications combines the structured and unstructured methods to achieve better efficiency and accuracy. Under this method, structured rectangular grids are used to cover most of the computational domain, and unstructured triangular grids are used only to patch between the rectangular grids (Fig. 1) or to conform to the curved boundaries of the computational domain (Fig. 2). In these figures, an unstructured triangular grid is used to accurately define the curved internal or external boundaries and a structured rectangular grid is used to decompose the regions of the computational domain that have a simple geometry.

Mathematical Model

We consider a system of two-dimensional Euler equations written in conservation law form:

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = 0 \tag{1}$$

where

$$U = \begin{vmatrix} \rho \\ \rho u \\ \rho v \\ e \end{vmatrix}, \quad F = \begin{vmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ u(e+p) \end{vmatrix}, \quad G = \begin{vmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ v(e+p) \end{vmatrix}.$$

Here u, v are the x, y velocity vector components, p is the pressure, ρ is the density, and e is total energy of the fluid.

The equation of state for water was adopted from Ref. 2. The actual pressure and density $\tilde{p}, \tilde{\rho}$ in water are modified and then applied in the Euler solver. The modified pressure and density are given as

$$p = \tilde{p} + B, \tag{2}$$

$$\rho = \tilde{\rho}/(1 + \tilde{p}/B)^n, \tag{2a}$$

where B = 2955 bar and n = 7.44 to adjust the velocity of sound to that for water (a_0 = 1483 m/sec).

The initial pressure distribution $\tilde{p}(r)$ in the left focal point is chosen as

$$\tilde{p}(r) = 1.0 \text{ bar } + \Delta p \exp\left[-(r - r_0)/(a_0 \tau)\right], \tag{2b}$$

where Δp is the intensity of the blast, τ is a time scale and a_o is sound speed in water ($\tau = 3\mu$sec).

It is assumed that an initial distribution of the fluid parameters is given at $t = 0$, and the boundary conditions defining a unique solution are specified for the computational domain.

Integration Algorithm

The system of governing equations (1) can be written in the following form:

$$\frac{\partial U}{\partial t} + \nabla \cdot \mathbf{Q} = 0, \tag{3}$$

where Q represents the convective flux vector. By integrating Eq. (3) over space and using Gauss' theorem, we obtain the following expression:

$$\frac{\partial}{\partial t}\int_{\Omega} U dA + \oint_{\partial\Omega} \mathbf{Q} \cdot d\mathbf{l} = 0, \tag{4}$$

where $d\mathbf{l} = \mathbf{n}dl$, $\mathbf{n}$ is the unit normal vector in the outward direction, and d_l is the element of length on the boundary of the domain. The variable Ω is the domain of computation and $\partial\Omega$ is the domain boundary.

Equation (4) can be discretized for each element (cell-triangle) of the domain:

$$\frac{(U_i^{n+1} - U_i^n)}{\Delta t} A_i = \sum_{j=1}^{3} \mathbf{Q}_j^{n+\frac{1}{2}} \mathbf{n}_j \Delta l_j, \tag{5}$$

where A_i is the area of the cell; Δt is the marching time step; U_i^{n+1} and U_i^n are the primitive variables at the center of the cell at time n and at the updated $(n+1)st$ timestep; $\mathbf{Q}_j^{n+\frac{1}{2}}$ are the value of the fluxes across the three boundaries edges on the circumference of the cell, where $\mathbf{n}_j$ is the unit normal vector to edge j of the boundary, and Δl_j is the length of the boundary edge

j. Equation (5) is used to update the physical primitive variables U_i according to computed fluxes for each timestep Δt. The time step is subjected to the Courant-Fredrichs-Levy (CFL) constraint.

To ensure a second order spatial accuracy, the gradient of each primitive variable is computed in the baricenter of the cell. This gradient is used to define the projected values of primitive variables at the two sides of the cell edge, as shown in Fig. 3. The gradient is approximated by a path integral

$$\int_{\Omega} \nabla U_i^{\text{cell}} dA = \oint_{\partial\Omega} U_j^{\text{edge}} d\mathbf{l} . \tag{6}$$

The notation is similar to the one used for Eq. (5), except that the domain Ω is a single cell and U_i^{cell} and U_j^{edge} are values at the baricenter and on the edge respectively. The gradient is estimated as

$$\nabla U_i^{\text{cell}} = \frac{1}{A}\sum_{j=1}^{3} \bar{U}_j^{\text{edge}} \mathbf{n}_j \Delta l_j , \tag{7}$$

where $\bar{U}_j^{\text{edge}}$ is an average value representing the primitive variable value for edge j.

The gradients that are computed at each baricenter are used to project values for the two sides of each edge by piecewise linear interpolation. The interpolated values are subjected to monotonicity constraints.[3] The monotonicity constraint assures that the interpolated values are not creating new extrema.

The monotonicity limiter algorithm can be written in the following form:

$$U_{\text{pro}}^{\text{edge}} = U_i^{\text{cell}} + \phi \nabla U_i \cdot \Delta \mathbf{r}, \tag{8}$$

where Δr is the vector from the baricenter to the point of intersection of the edge with the line connecting the baricenters of the cells over the two sides of this edge. ϕ is the limiter coefficient that limits the gradient ∇U_i.

First, we compute the maximum and minimum values of the primitive variable in the i's cell and its three neighboring cells that share common edges (see Fig. 3):

$$\left.\begin{aligned} U_{\text{cell}}^{\max} &= \max\left(U_k^{\text{cell}}\right) \\ U_{\text{cell}}^{\min} &= \min\left(U_k^{\text{cell}}\right) \end{aligned}\right\} k = i, 1, 2, 3 . \tag{9}$$

The limiter can be defined as:

$$\phi = \min\left\{1, \phi_k^{lr}\right\}, \; k = 1, 2, 3, \tag{10}$$

where the superscript lr stands for left and right of the three edges (6 combinations altogether). ϕ_k^{lr} is defined by:

$$\phi_k^{lr} = \frac{\left[1 + \mathrm{Sgn}\left(\Delta U_k^{lr}\right)\right] \Delta U_{\text{cell}}^{\max} + \left[1 - \mathrm{Sgn}\left(\Delta U_k^{lr}\right)\right] \Delta U_{\text{cell}}^{\min}}{2\Delta U_k^{lr}}$$

$$k = 1, 2, 3, \tag{11}$$

where $\Delta U_k^{lr} = \nabla U_i^{lr} \cdot \Delta r_k$ and

$$\left.\begin{aligned} \Delta U_{\text{cell}}^{\max} &= U_{\text{cell}}^{\max} - U_i^{\text{cell}} \\ \Delta U_{\text{cell}}^{\min} &= U_{\text{cell}}^{\min} - U_i^{\text{cell}} \end{aligned}\right\} . \tag{12}$$

To obtain second-order accuracy in space and time, we subject the projected values of the left and right side of the cell edge to characteristic constraints following Ref. 4. The one-dimensional characteristic predictor is applied to the projected values at the half timestep $t^n + \Delta t/2$. The characteristic predictor is formulated in the local system of coordinates for the one dimensional Euler equation. We illustrate the implementation of the characteristic predictor in the direction of the unit vector $\mathbf{n}_c$. The Euler equations for this direction can be written

$$W_t + A(W) W_{nc} = 0, \tag{13}$$

where

$$W = \begin{Bmatrix} \tau \\ u \\ p \end{Bmatrix}; \; A(W) = \begin{pmatrix} u & -\tau & 0 \\ 0 & u & \tau \\ 0 & \rho c^2 & u \end{pmatrix}, \tag{14}$$

where $\tau = \rho^{-1}$, ρ denotes density, and u, p are the velocity and pressure. The matrix $A(W)$ has three eigenvectors $(l^{\#}, r^{\#})$ (l for left and r for right, where $\#$ denote +,0,−) associated with the eigenvalues $\lambda^+ = u + c$, $\lambda^0 = u$, $\lambda^- = u - c$.

An approximation of the value projected to an edge, accurate to second order in space and time, can be written

$$\begin{aligned} W_{i+\Delta r}^{n+1/2} &\approx W_i^n + \frac{\Delta t}{2}\frac{\partial W}{\partial t} + \Delta r \frac{\partial W}{\partial r_{nc}} \\ &\approx W_i^n + \left[\Delta r - \frac{\Delta t}{2} A(W_i)\right] \frac{\partial W}{\partial r_{nc}} \end{aligned} \tag{15}$$

An approximation for $W_{i+\Delta r}^{n+1/2}$ can be written as

$$W_{i+\Delta r}^{n+1/2} = W_i + (\Delta \mathbf{r}_i - \frac{\Delta t}{2}(M_x M_n)\mathbf{n}_c).\nabla W_i, \tag{16}$$

where

$$(M_x M_n) = \begin{cases} \mathrm{Max}(\lambda_i^+, 0) & \text{for cell left to the edge} \\ \mathrm{Min}(\lambda_i^-, 0) & \text{for cell right to the edge .} \end{cases} \tag{17}$$

The gradients ied in the process of computing the projected values at $t^n + \Delta t/2$ are subjected to the monotonicity limiter.

Following the characteristic predictor described above, the full Riemann problem is solved at the edge. The solution of the Riemann problem defines the flux $Q_j^{n+\frac{1}{2}}$ through the edge. The fluxes through the edges of triangles are then integrated (Eq. 5), thus updates the variables at t^{n+1}. One of the advantages of this algorithm is that calculation of the fluxes is done over the largest loop in the system (the loop over edges) and can be carried out in the vectorized or parallelized loop. This makes the algorithm efficient.

The algorithm presented is a modification of the algorithm of Ref. 5, which was derived for a structured mesh. The present algorithm has been applied to simulate a wide range of flow problems and has been found to be very accurate in predicting the features of the physics. The performance of the algorithm is well documented in Refs. 6-9. The algorithm for the rectangular cells are identical except the cell has four edges (Eq. 5).

Sound Wave Focusing in an Ellipsoidal Reflector

For our simulations, we chose a deep reflector shaped like an ellipsoid, which was used for ESWL by Dornier and other companies. A schematic of the cross section of this reflector is shown in Fig. 4. Strong acoustic waves are generated in the left focal point of the ellipsoid by an instantaneous release of energy and are refocused at the right focal point. Ideally, a reflector should employ waves of acoustic intensity, since the nonlinear reflections of strong shock waves lead to significant distortions in wave propagation and impair simple geometrical focusing.

Figure 2 shows the computational domain and grid for the ellipsoidal reflector that we used in our study. In order to illustrate the concept of the composite structured/unstructured grid, we have shown only every sixteenth cell of the grid that was actually used for the simulation. In this example, we observe that the structured rectangular grid covers about 90% of the computational domain, and the unstructured triangular grid is restricted to the curved surface of the ellipsoid and covers about 10% of the domain. The major axis of the ellipsoid is 150 mm and the minor axis is 90 mm.

Two simulations were conducted with two different Δp values to study how the intensity of the blast affects focusing of waves in the reflector. The first simulation was done with $\Delta p = 725$ bar and $\tau = 3\mu s$ where $|r - r_0| <$ 10 mm. The other simulation was done by using pressure three times larger than in first simulation.

In Figs. 5a-5d simulation results for the $\Delta p = 725$ bar conditions are shown in the form of pressure contour plots. Figure 5a shows pressure distribution for the initial stage of wave propagation before the wave front has reached the surface of the reflector. The contour plots are shown at t=1.10 ×10 sec. At this time the maximum pressure in the wave has dropped to 173 bar. In Fig. 5b pressure contours are shown at t=3.32 $\times 10^{-5}$sec. Here we observe that the wave reflected from the surface of the reflector has maximum pressure about five times than that of the incident wave. However, both wave fronts propagate through the water with a constant speed equal to the speed of sound, and the phase shift observed in Fig. 5b holds through the calculation. In Fig. 5c the simulation results are shown at the stage when the incident wave is crossing the center of symmetry of the reflector. Here $t = 8.88 \times 10^{-6}$sec. It is interesting to note that the value of the overpressure at this location was used in Ref. 1 as a normalizing value for presentation of the experimental and computational results. In our case for the initialization with $\Delta p = 725$ bar the incident pressure at the center of the ellipsoid is $p = 11.1$ bar. In Fig. 5d simulation results are shown at $t = 19.2 \times 10^{-5}$sec, when maximum focusing of the reflected wave take place. The pressure values in the focal point reaches 188 bar. This maximum is immediately followed by a negative phase with a minimal pressure of 163 bar. This strong pressure variations can cause disintegration of the stones by the ESWL apparatus.

In Figs. 6a-6b simulation results are shown for the second case of $\Delta p = 2175$ bar. As we can see in Fig. 6a, this value of the initial overpressure produces an incident wave with about 33 bar, which is a bit higher than the 29 bar value observed in Ref. 1. The wave structure at the time of focusing is shown in Fig. 6b. Here we can observe that for this case the maximum pressure reaches 494 bar, followed by a 371 bar minimum. Comparing this case with that reported above, we conclude that the amplification at the focal point is smaller in the second case.

The waves observed in the system are of acoustic intensity and are propagating at the speed of sound. The reflected wave will therefore not be able to catch up with the incident wave. Except for some compressibility effects in the initiation and focusing stages when pressures are high, the fluid will behave as incompressible. Figure 7 shows the density contour for the first case ($\Delta p = 725$ bar). As expected, the compressibility effect is negligible.

In Fig. 8 the simulation results are compared with the experimental results in a plot of normalized pressures as function of distance from the focal point. In this figure the simulation results for the case of initiation with $\Delta p = 725$ bar and $\Delta p = 2175$ bar are shown by the curves marked with triangles and rectangles respectively. The experimental results for the 29-bar incident pressure

(which most closely fits our second simulation) are shown by the curve marked by circles. In Fig. 8 we see that the maximum reflection factor is achieved for weaker waves, which is consistent with the results reported in Ref. 1. The simulation results are very close to the experimental ones in the case of $\Delta p = 2175$ bar initiation for focal point location and pressure amplification factor, which validates the simulation methodology.

In all the figures presented, the method of composite domain decomposition works extremely well, producing solutions with no seams at the interfaces. We should mention here that our test problem is particularly sensitive because the main acoustic waves are weak, and any inaccuracy introduced at the grid interfaces would produce a distortion in the phase or in the intensity of the traveling waves that would be a visible disturbance evident in the results.

Conclusions

A composite method of structured/unstructured domain decomposition is introduced as an efficient technique for dealing with the computational domains of complex geometry. We have simulated a demanding acoustic wave focusing problem and have shown that our approach leads to accurate wave propagation without any reflection or distortion at the structured/unstructured grid interfaces. Note that for the acoustic focusing problem as simulated and presented in this paper, both structured and unstructured methods of domain decomposition can be shown to be inadequate if used separately. The structured method has difficulty describing the curved boundaries of the computational domain, while the unstructured method is totally inefficient in describing phenomena with wide fronts that occupy a large portion of the computational domain. Our hybrid method combines the advantages of structured and unstructured methods of domain decomposition. This hybrid technique combines the efficiency of the unstructured grid, which accurately represents curved walls, with the computational and memory efficiency of the structured grid in the majority of the computational domain. We also attribute the quality of the numerical result to the Second Order Godunov method, which allows a consistent, accurate and robust formulation for handling both grids and boundary conditions.

Acknowledgment

The work reported here was partially supported by ARPA and ONR. The authors would like to thank Col James Crowley and Dr. Lawrence Kabecoff for their interest in this project.

References

1. H. Gronig, "Past, Present and Future of the Shock Focusing Research," Proceedings of the International Workshop on Shock Wave Focusing, Sendai, Japan, March 1989.
2. K. Isuzugawa and M. Horiuchi, "Experimental and Numerical Studies of Blast Wave Focusing in Water," Proceedings of the International Workshop on Shock Wave Focusing, Sendai, Japan, March 1989.
3. B. van Leer, "Towards the Ultimate Conservative Difference Scheme, V.A. Second Order Sequel to Godunov's Method," J. Comp. Phys. 32, 101-136 (1979).
4. P. Collela and P. Woodward, "The Piecewise Parabolic Method (PPM) for Gasdynamic Simulations," J. Comp. Phys. 54, 174-201 (1984).
5. S. Eidelman, P. Collela, and R.P. Shreeve, "Application of the Godunov Method and Its Second Order Extension to Cascade Flow Modeling," AIAA Journal 22, 10, 1984.
6. I. Lottati, S. Eidelman and A. Drobot, "A Fast Unstructured Grid Second Order Godunov Solver (FUGGS)," 28th Aerospace Sciences Meeting, AIAA-90-0699, Reno, NV (1990).
7. I. Lottati, S. Eidelman, and A. Drobot, "Solution of Euler's Equations on Adaptive Grids Using a Fast Unstructured Grid Second Order Godunov Solver," Proceeding of the Free Lagrange Conference, Jackson Lake, WY, June 1990.
8. I. Lottati and S. Eidelman, "Second Order Godunov Solver on Adaptive Unstructured Grids," Proceeding of the 4th International Symposium on Computational Fluid Dynamics, Davis, CA, September 1991.
9. I. Lottati and S. Eidelman, "A Second Order Godunov Scheme on Spatial Adapted Triangular Grid," To appear in a special issue of Applied Numerical Mathematics (Proceedings of U.S. Army Workshop on Adaptive Methods for Partial Differential Equations, R.P.I., March 1992).

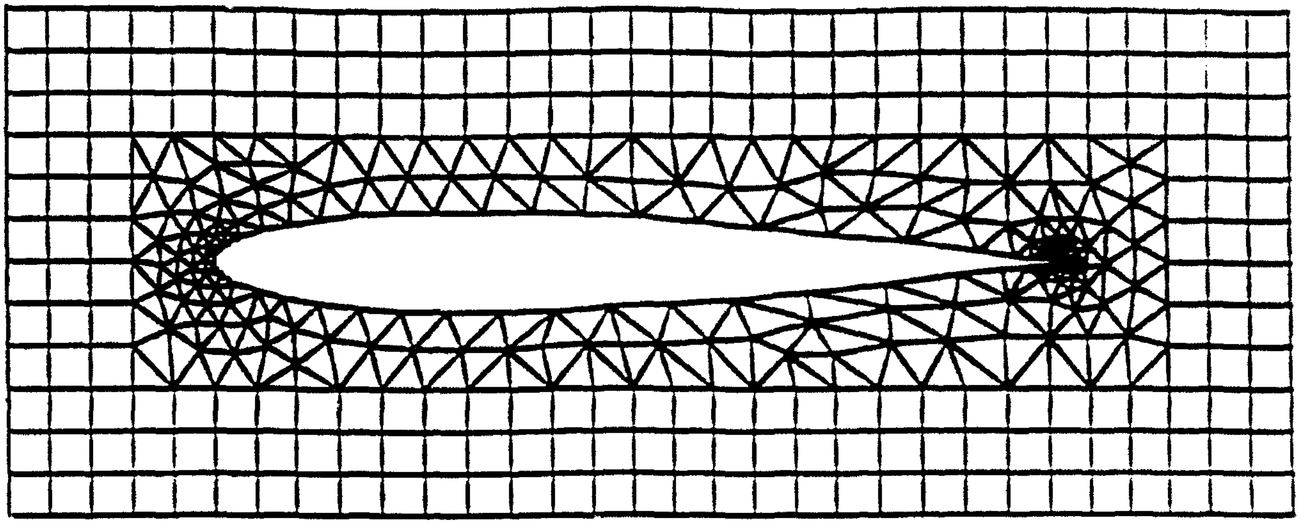


Figure 1. A possible candidate configuration for hybrid structured/unstructured domain decomposition.

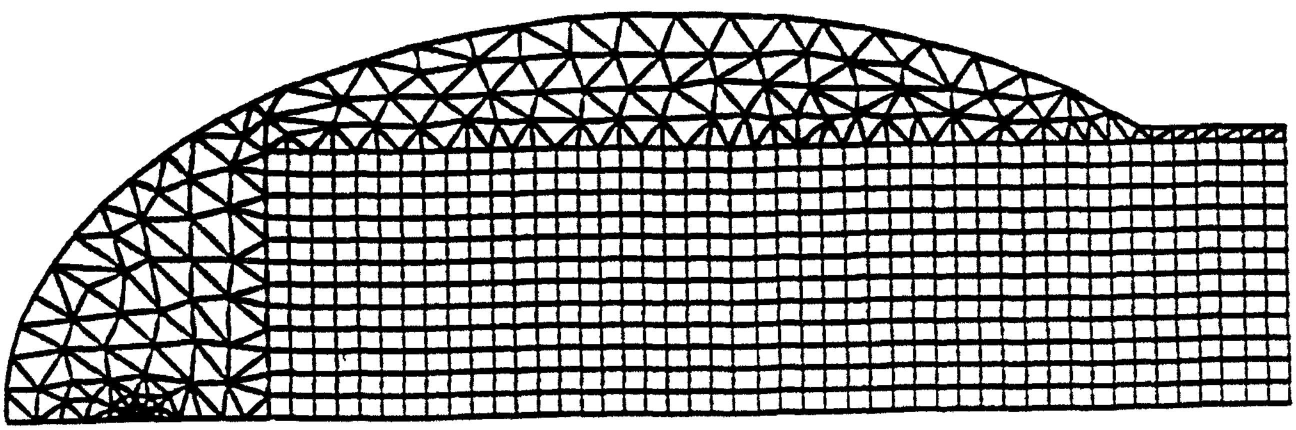


Figure 2. A possible candidate configuration for hybrid structured/unstructured domain decomposition, representing the ellipsoid reflector grid used for the numerical simulation.

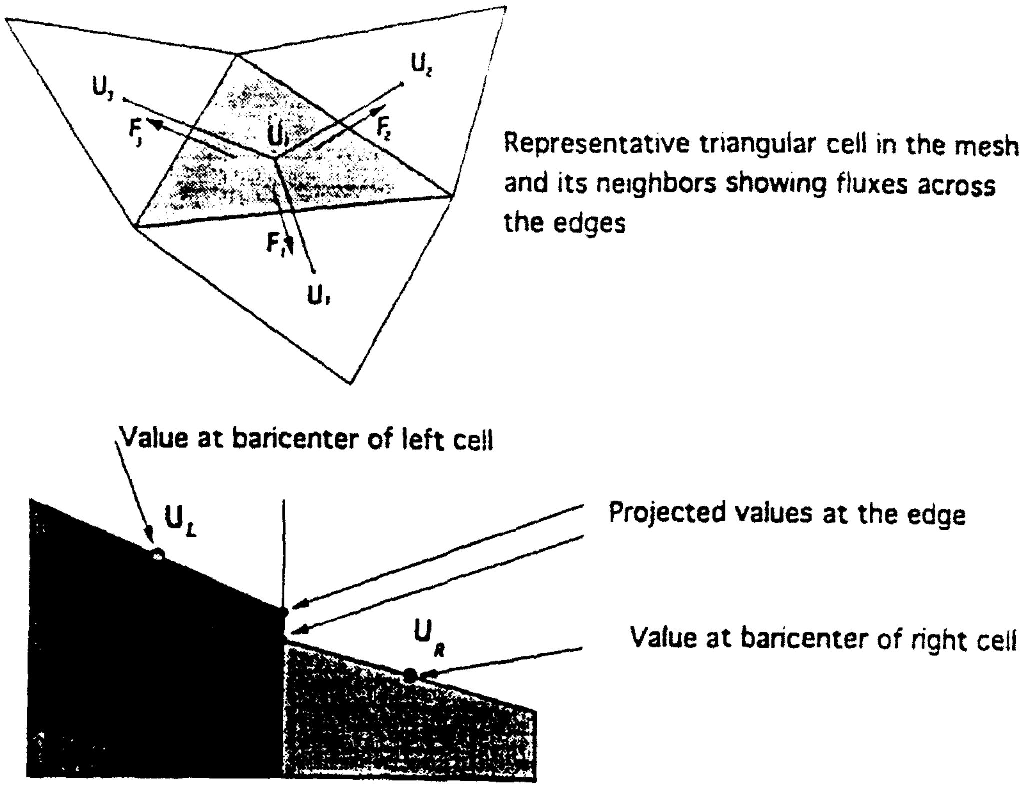


Figure 3. Second order triangular based flux calculation.

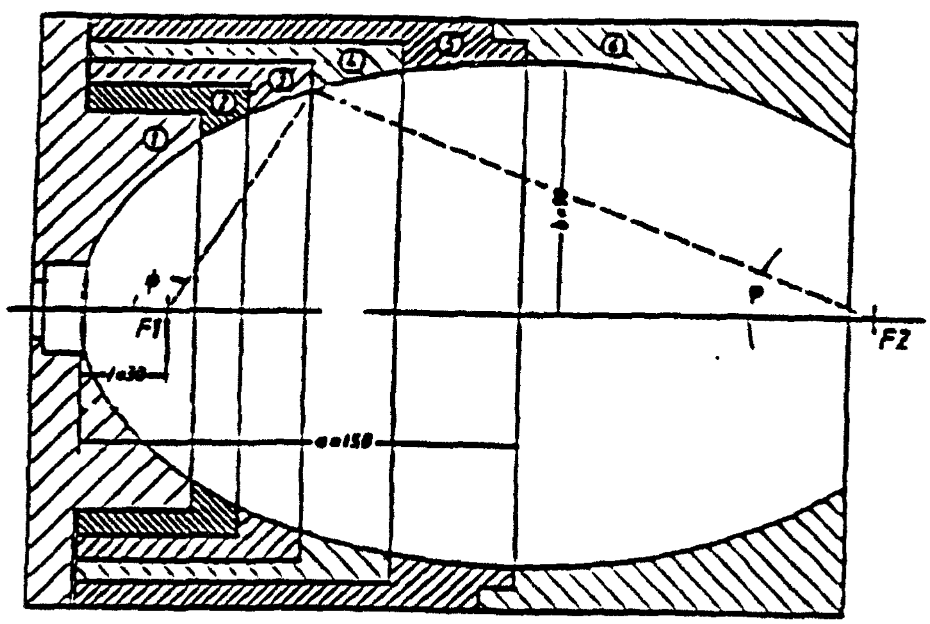
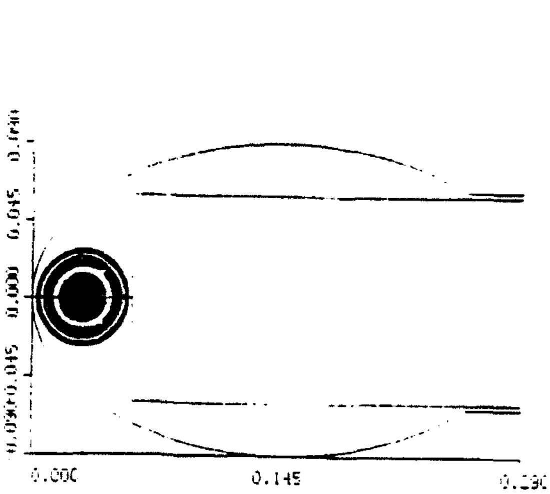
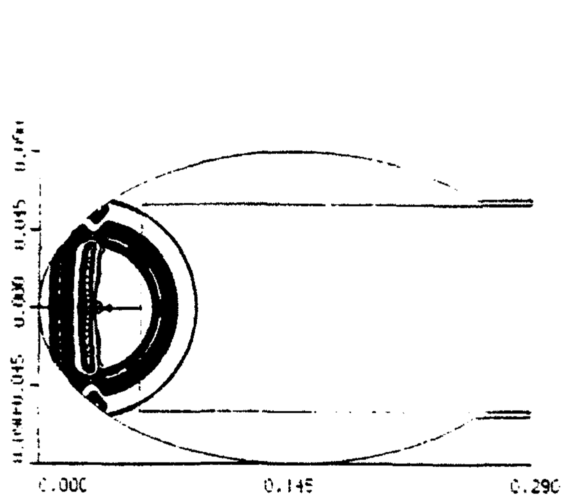


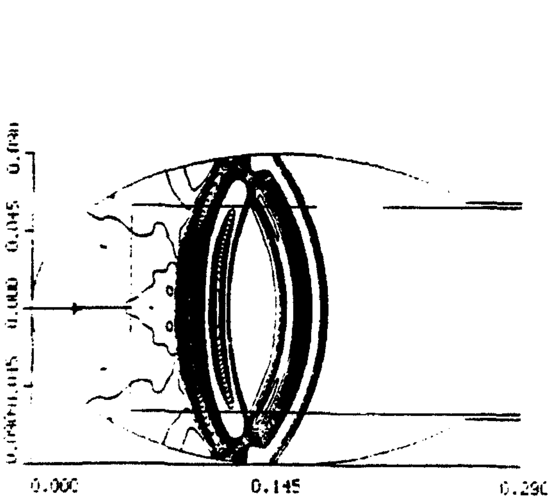
Figure 4. A schematic drawing of the center cross section of the ellipsoid reflector.



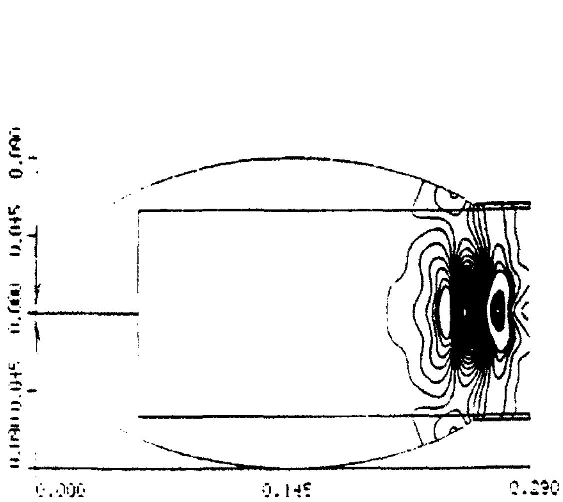
a. Time = 1.10 $\times 10^{-5}$sec



b. Time = 3.32 $\times 10^{-5}$sec

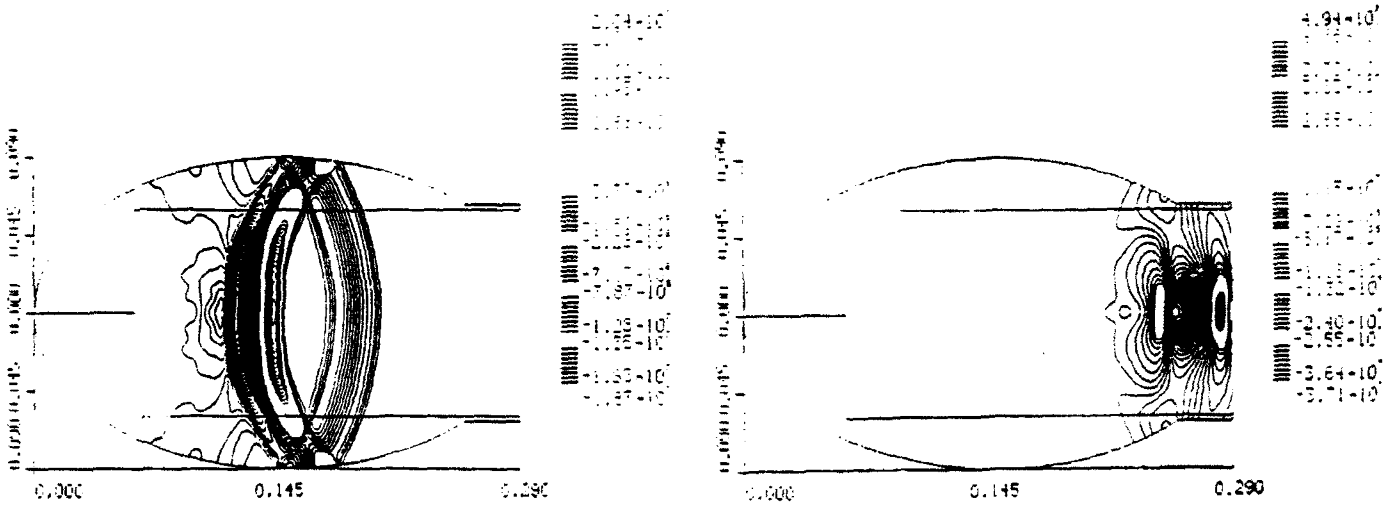


c. Time = 8.88 $\times 10^{-5}$sec



d. Time = 1.92 $\times 10^{-4}$sec

Figure 5. Pressure contours showing the incident wave and the reflected wave pattern for Δp = 725 bar.



a. Time = 1.08×10^{-4}sec

b. Time = 1.96×10^{-4}sec

Figure 6. Pressure contours showing the incident wave and the reflected wave pattern for $\Delta p = 2175$ bar.

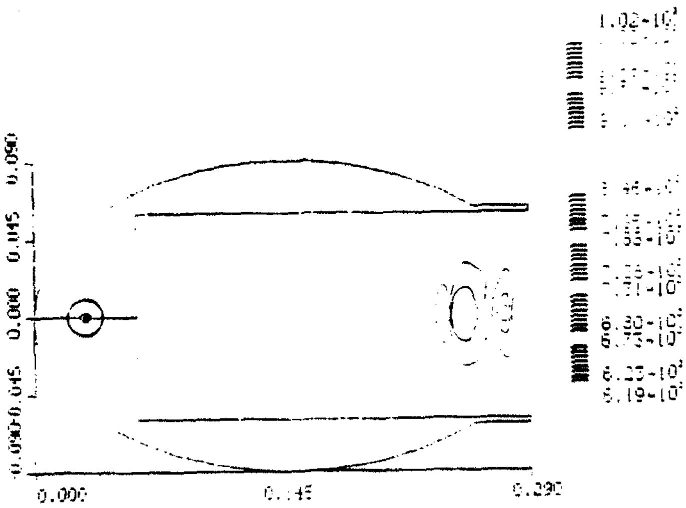


Figure 7. Density contours emphasizing the fact that the compressibility effect is negligible ($\Delta p = 725$ bar at $t = 1.92 \times 10^{-4}$sec).

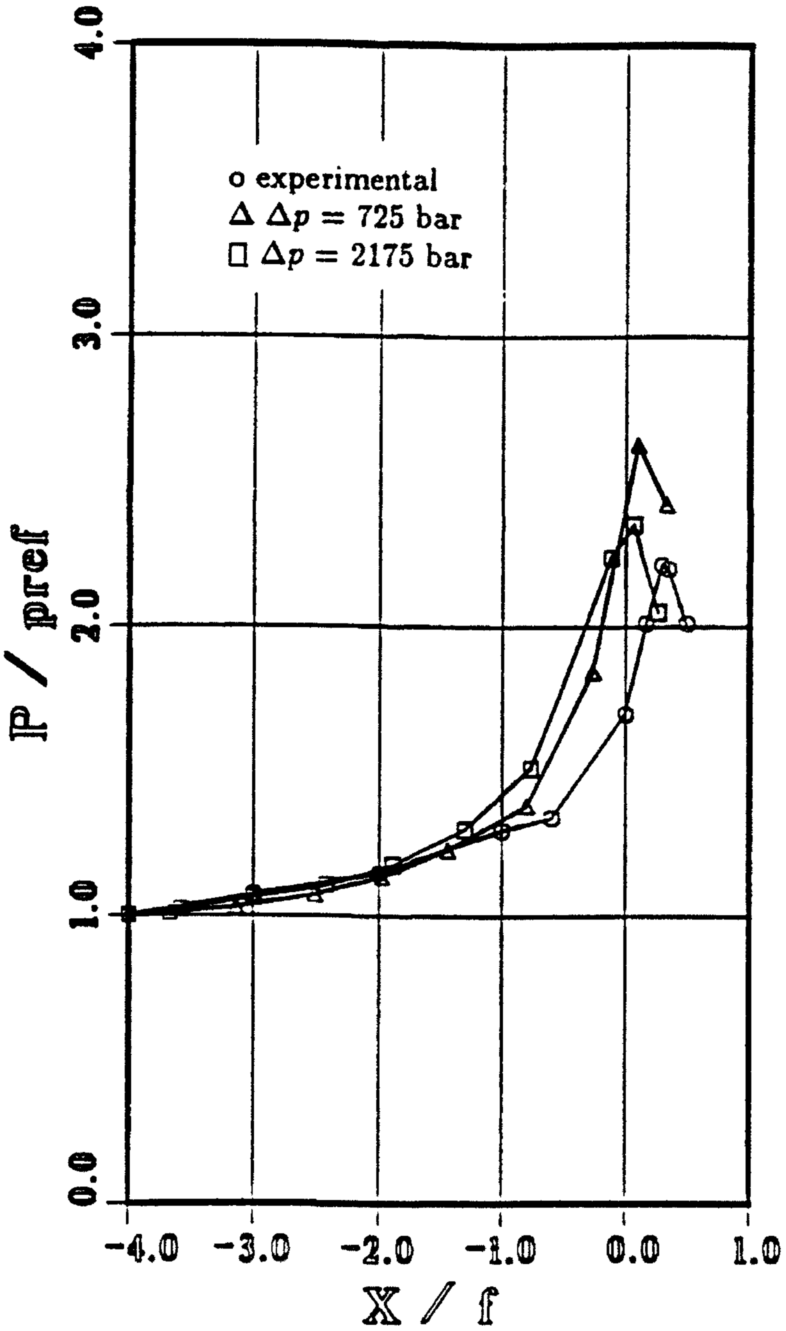


Figure 8. Normalized maximum pressure distribution on the axis of symmetry. A comparison between computed and experimental results.